# 1. Fundamentação teórica

Neste capítulo serão abordados os conceitos necessários para melhor compreensão e aplicação dos assuntos dos capítulos posteriores, que dizem respeito à implementação de banco relacionais. Um bom profissional da área de banco de dados precisa conhecer os três níveis envolvidos em um projeto de bancos de dados: conceitual, lógico e físico. Nesta disciplina será abordado basicamente o terceiro nível, no entanto, o conhecimento dos níveis físico e lógico é pré-requisito para um bom implementador.

## 1.1. Banco de Dados

Podemos entender como Banco de Dados(Data Base) qualquer sistema que reúna e mantenha organizada uma série de informações relacionadas a um determinado assunto, em uma determinada ordem.
Um sistema de manutenção de informações por computador, consideradas como significativa ao indivíduo ou a organização servida pelo sistema, cujo objetivo global é manter as informações atualizadas e torná-las disponíveis quando solicitadas para o processo de tomada de decisão.
O objetivo principal de um Banco de Dados é fornecer um ambiente que seja adequado e eficiente para o uso na recuperação e no armazenamento da informação.
Uma agenda telefônica, catálogo que fica ao lado do aparelho telefônico, não deixa de ser um banco de dados: nela temos telefones de várias pessoas de uma região geográfica, organizados pelo sobrenome. O computador é apenas uma ferramenta eficiente para montar um banco de dados, pois nele inserimos muitas informações e as localizamos de forma extremamente rápida.

Os bancos de dados implementados em computadores são divididos em parte Lógica e parte Física:

**A parte física** é constituída dos equipamentos de hardware necessários para o armazenamento dos dados, processamento das informações e acesso aos dados, como por exemplo: processadores, memórias, HDs, dispositivos de entrada e saída, enfim são os computadores (servidores e clientes).
**A parte lógica** é o conjunto de softwares que possibilitam a implementação do modelo projetado, o controle de acesso aos usuários, a compilação das instruções, enfim é o Sistema Gerenciador do Banco de Dados.

Já a concepção de um banco de dados, compreendida pela parte lógica do sistema de banco de dados, se dá em três níveis: conceitual, lógico e físico:

**Nível conceitual**: consiste na identificação dos dados que deverão ser armazenados e no entendimento de como estes dados serão utilizados para

produção de informações[1]. Neste nível não são, necessariamente, consideradas particularidades relacionadas ao modelo de banco de dados que será utilizado para a implementação do banco. É a etapa de entendimento do negócio e sua representação.
No contexto de desenvolvimento de projetos, este nível está associado a etapa de análise de requisitos.
**Nível lógico:** consiste na representação de como os dados serão armazenados seguindo uma metodologia. De acordo com Cougo[1] essa representação é independente dos dispositivos ou meios de armazenamento físico das estruturas de dados, mas deve ser elaborado respeitando-se os conceitos tais como chaves de acesso, controles de chaves duplicadas, normalização, integridade referencial entre outros.
No contexto de desenvolvimento de projetos, este nível está associado à etapa de projeto.
**Nível Físico:** Consiste na implementação das estruturas de dados, dos relacionamentos, das chaves, enfim é a etapa da construção do banco de dados considerando os aspectos físicos e as características do sistema gerenciador de bancos de dados utilizado.
No contexto de desenvolvimento de projetos, este nível está associado à etapa de implementação ou operação.

## 1.2. Bancos de dados relacionais

É conjunto de dados sobre um "negócio" organizados por assunto, onde cada assunto é representado por uma tabela, por exemplo: uma pizzaria precisa de informações sobre: os fornecedores dos produtos que utiliza, o estoque de seus produtos, clientes, pizzas etc., sendo assim podemos separar as os dados que representarão as informações de fornecedores, clientes, produtos, etc.. Nos modelos relacionais estes assuntos são representados por tabelas.
As tabelas são compostas por conjuntos de campos, denominados registros (tuplas), que armazenam dados.
Os dados existem fisicamente e precisam de um contexto para adquirir algum significado. São estáticos, isto é, permanecem no mesmo estado até que sejam modificados por um processo manual ou automático.

| Mellanzone | 55 | 03909-70 | 22,00 | Mesozóica |
|---|---|---|---|---|

Isoladamente estes dados não têm nenhum sentido. O que é Mellanzone? Será o nome de uma pessoa, de um supermercado ou de uma pizza? E 55? É um código? Uma soma? Uma nota?

[1] Alguns autores não fazem distinção entre dados e informações, mas para evitar interpretações trataremos dado como uma valor fisicamente registrado no banco de dados e informação como o significado atribuído a estes valores de acordo com o seu contexto de utilização.

Outra característica dos dados é que eles são estáticos, ou seja, permanecem no mesmo estado até que sejam modificados por algum processo manual ou automático.
Os dados tornam-se informações quando são associados a um contexto e transmitem significados lógicos às pessoas.

| Nome da Pizza | Ingredientes | Código Pizza | Preço da pizzza | Apelido da Pizza |
|---|---|---|---|---|
| Mellanzone | 55 | 03909-70 | 22,00 | Mesozóica |

Os dados são armazenados em campos. Um campo é a menor estrutura dentro de um banco de dados relacional. Cada campo possui um conjunto de características tais como identificador do campo, tipo de dado que será armazenado, tamanho do dado e restrições[2], por exemplo:

| Campo | Identificador do campo | Tipo de dado | Tamanho | Restrições |
|---|---|---|---|---|
| Nome da Pizza | nome_pizza | alfanumérico | 20 | preenchimento obrigatório |
| Ingredientes | ingredientes | numérico inteiro | 2 | preenchimento obrigatório, chave estrangeira |
| Código da Pizza | codigo_pizza | numérico inteiro | 8 | chave primária |
| Preço da pizza | preco | numérico real | 5 | nenhuma |
| Apelido da pizza | apelido | alfanumérico | 15 | não pode ser repetido em outro registro. |

O conjunto de campos sobre um assunto compõe um registro. Um registro equivale a uma linha de
uma tabela e também é conhecido como tupla, por exemplo:

**Tabela: CARDÁPIO**

| Nome da Pizza | Ingredientes | Código | Preço | Apelido |
|---|---|---|---|---|
| Mellanzone | 55 | 03909-70 | 22,00 | Mesozóica |
| Tomate Seco | 23 | 02983-89 | 21,00 | Pomodori |
| Calabresa | 11 | 19203-89 | 11,00 | Kashu |

A tabela CARDÁPIO contém:
Os campos: Nome da Pizza, Ingredientes, Código, Preço e Apelido;
O campo Nome da Pizza recebeu os dados: Mellazone, Tomate Seco e Calabresa;
O 3º. registro é composto pelo conjunto de dados: Calabresa, 11, 19203-89, 11,00 e Kashu;

---

[2]A especificação destas características deve seguir os padrões do SGBDR escolhido

Cada linha da tabela representa uma tupla ou registro;
Cada coluna da tabela representa um campo.

Os campos podem ainda, ser classificados como composto, multivalorado ou calculado e devem ser implementados de acordo com o modelo projetado que deve atender às especificidades do negócio. A seguir serão apresentadas as características de cada um:

- **Campo de múltiplas partes ou campo composto:** armazena dados com valores distintos. No exemplo abaixo, o campo endereço possui informações sobre o nome da rua (alfanumérico) e o número do imóvel (numérico).

| Instrutor | Endereço |
|---|---|
| Lúcia da Silva Pires | Rua Sebastião Marchesone, 500 |

**Campo Simples** **Campo contendo múltipas partes**

- **Campo multivalorado**: armazena múltiplas instâncias de um mesmo tipo, no exemplo abaixo a instrutora Lúcia ensina diversas categorias: Cisco, Oracle, Asp, neste caso o campo Categorias ensinadas é multivalorado pois recebe muitos dados sobre o mesmo assunto para uma única instrutora.

| Instrutor | Categorias ensinadas |
|---|---|
| Lúcia da Silva Pires | Cisco, Oracle, Asp |

**Campo Simples** **Campo multivalorado**

- **Campo calculado**: armazena um resultado de uma expressão matemática, no exemplo abaixo o campo Total é o resultado da multiplicação entre os campos Quantidade e Preço Unitário:

| Quantidade | Preço Unitário | Total |
|---|---|---|
| 10 | 12,00 | 120,00 |
| **Campos Simples** | | **Campo calculado** |

O projeto de um banco de dados relacional consiste em um conjunto de tabelas que devem estar relacionadas de maneira que não exista a necessidade de redundâncias no armazenamento de dados. Os relacionamentos entre as tabelas são estabelecidos por meio dos campos chave primária de uma tabela com um campo chave estrangeira de outra tabela.

**ATENÇÃO:**

- **Chave Primária:** É um campo ou um conjunto de campos que identifica unicamente cada registro da tabela, sendo assim, um registro náo pode conter, neste campo, um dado que já esteja armazenado em outro registro.

- **Chave Estrangeira:** É um campo ou conjunto de campos usado para estabelecer um relacionamento entre duas tabelas. Na tabela de origem deve ser chave primária.

No exemplo abaixo, as tabelas EMP e DEPT[3] contém, respectivamente, informações sobre os funcionários e sobre os departamentos de uma empresa e para que não exista necessidade de armazenar os dados sobre o nome do departamento e a localização dos mesmos para cada registro de funcionários elas são relacionadas por meio dos campos DEPTNO que na tabela EMP é chave estrangeira e na tabela DEPT é chave primária.

**Tabela EMP**

| Empno | Ename | Deptno |
|---|---|---|
| 7839 | KING | 10 |
| 7782 | CLARK | 10 |
| 7566 | JONES | 20 |
| 7698 | BLAKE | 30 |
| | | **PK[4]** |

**Tabela DEPT**

| Deptno | Dname | Loc |
|---|---|---|
| 10 | ACCOUTING | NEW YORK |
| 20 | RESEARCH | DALLAS |
| 30 | SALES | CHICAGO |
| 40 | OPERATIONS | BOSTON |
| **FK[5]** | | |

Como os dados sobre entidades diferentes são armazenados em tabelas diferentes, talvez você precise combinar duas ou mais tabelas para responder a uma pergunta específica. Por exemplo, talvez seja necessário saber a localização do departamento no qual um funcionário trabalha. Nesse cenário, você precisa de informações da tabela EMP (que contém dados sobre funcionários) e da tabela DEPT (que contém informações sobre departamentos).
O recurso de relacionar dados de uma tabela a dados de outra permite organizar informações em unidades gerenciáveis separadas. É possível manter logicamente os dados dos funcionários separadas dos dados dos departamentos armazenando-os em uma tabela separada.

---

[3] As tabelas emp e dept bem como os dados nela inseridos fazem parte de um banco de dados exemplo disponível no Oracle.
[4] PK – Primary Key – Chave Primária
[5] FK Foreing Key – Chave Estrangeira

## 1.3. Sistemas Gerenciadores de Bancos de Dados

De acordo com Silberschatz[2], um SGBD é constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados tendo como objetivo proporcionar um ambiente tanto conveniente quanto eficiente para a recuperação de dados.
O conjunto de dados contém informações sobre um assunto em particular, este assunto pode ser uma empresa, um projeto, um negócio entre outros.
O SGBDR ou RDBMS (*Relational Database Management System)* são sistemas que disponibilizam recursos para implementação de projetos de bancos de dados relacionais, manutenção e administração de suas estruturas, dados e usuários.
Com conceito de centralizar os dados e torná-los disponíveis a vários usuários conseguimos incrementar a velocidade, ter os dados centralizados, também torna mais fácil a administração e a segurança do banco de dados. Este conceito é chamado de SGBDR cliente/servidor. Neste tipo de SGBDR, os dados ficam fisicamente armazenados em um computador que age como um servidor de banco de dados e os usuários interagem com esse banco por meio de aplicações localizadas em seus próprios computadores denominados clientes. Estes sistemas são amplamente utilizados para administrar grandes volumes de dados compartilhados.
Atualmente podem ser encontrados diversos SGBDR para quase todos os sistemas operacionais e que podem ser usados para atender inúmeras necessidades. Alguns exemplos de sistemas gerenciadores de bancos de dados relacionais são: Oracle, Sybase, MySql, SQLServer, Access, entre outros.

De maneira geral os SGBDR apresentam algumas características operacionais desejáveis, são elas:

- **Controle de Redundâncias:** A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Por exemplo: Quando informações estão em locais diferentes, onde uma é atualizada e outra não. Em um Banco de Dados as informações só se encontram armazenadas em um único local, não existindo duplicação descontrolada dos dados. Quando existem replicações dos dados, estas são decorrentes do processo de armazenagem típica do ambiente Cliente-Servidor, totalmente sob controle do Banco de Dados.
- **Compartilhamento dos Dados**: deve possuir um sistema de controle de concorrência ao acesso dos dados, garantindo em qualquer tipo de situação a escrita/leitura de dados sem erros. Como exemplo imagine que um casal possui uma conta corrente conjunta cujo saldo é R$ 100,00, e ambos tentam efetuar um saque exatamente no valor de R$ 100,00, no mesmo momento e em locais diferentes.
- **Controle de Acesso:** deve possuir um sistema de controle dos privilégios de acesso aos dados cada um dos usuários ou grupos de usuários do banco.
- **Interfaceamento:** Formas de acesso gráfico, em linguagem natural, em SQL ou ainda via menus de acesso, não sendo uma "caixa-preta" somente sendo passível de ser acessada por aplicações.

- **Esquematização:** Mecanismos que possibilitem a compreensão do relacionamento existente entre as tabelas e de sua eventual manutenção.
- **Controle de Integridade:** Deve impedir que aplicações ou acessos pelas interfaces possam comprometer a integridade dos dados.
- **Backups e recuperação de dados:** Deve possuir opções para realização de cópias de segurança e recuperação de dados.

Além das características operacionais apresentadas anteriormente, o SGBD deve possuir alguns componentes para possibilitar a realização das suas tarefas:

- **Gerenciador de Acesso ao Disco:** O SGBD utiliza o Sistema Operacional para acessar os dados armazenados em disco, controlando o acesso concorrente às tabelas do Banco de Dados. O Gerenciador controla todas as pesquisas (queries) solicitadas pelos usuários, os acessos do compilador DML, os acessos feitos pelo Processador do Banco de Dados ao Dicionário de Dados e também aos próprios dados.
- **Compilador DDL (Data Definition Language):** Processa as definições do esquema do Banco de Dados, acessando quando necessário o Dicionário de Dados do Banco de Dados.
- **Dicionário de Dados:** Contém o esquema do Banco de Dados, suas tabelas, índices, forma de cesso e relacionamentos existentes.
- **Processador do Banco de Dados:** Manipula requisições a Base de Dados em tempo de execução. É o responsável pelas atualizações e integridade da Base de Dados.
- **Processador de Pesquisas (Queries dos usuários):** analisa as solicitações, e se estas forem consistentes, aciona o Processador do Banco de Dados para acesso efetivo aos dados.
- **Compilador DML (Data Manipulation Language):** As aplicações fazem seus acessos ao pré-compilador DML da linguagem hospedeira, que os envia ao Compilador DML (Data Manipulation Language) onde são gerados os códigos de acesso ao Banco de Dados.

Os Sistemas Gerenciadores de Bancos de Dados além de serem classificados de acordo com o modelo utilizado para o projeto e implementação também podem ser classificados de como mono-usuário, utilizado em computadores pessoais ou multi-usuários, por exemplo Cliente/servidor; localizado ou distribuído; se ele for localizado, então todos os dados ficam armazenados em um mesmo local (máquinas ou discos) no distribuído os dados ficam distribuídos em vários locais distintos e ainda um SGBD pode possuir um ambiente homogêneo é o ambiente composto por um único SGBD ou um ambiente heterogêneo compostos por diferentes SGBDs, este assunto será explorado com mais detalhes em outras disciplinas ao longo do curso.

## 1.4. OBJETOS DO BANCO DE DADOS.

A criação de um banco de dados compreende uma série de instruções para definição de como serão as características do banco; onde os dados serão fisicamente armazenados; quantos usuários irão utilizar o banco simultâneamente; como será a configuração da instância do banco, entre outras informações. Mas não basta apenas criar o banco é necessário que este banco tenha estruturas que possibilitem o armazenamento de dados, o acesso aos dados, mecânismos que auxiliem na recuperação dos dados, entre outros. Os objetos do banco representam estas estruturas e são criados e manipulados com as instruções DDL – Data Definition Language[6] e as suas definições ficam armazenadas no dicionário de dados (Data Dictionary) do banco. Como exemplos de objetos pode-se citar:

- Tabela (Table)
- Visão (View)
- Sequencia (Sequence)
- Índice (Index)
- Usuário (User)
- Personagem (role)
- Procedimentos (Procedure)
- Funções (Function)
- Gatilhos (Trigger)
- Pacotes (Package)

## 1.5. Usuários do Banco de Dados

Os usuários do banco de dados são as pessoas que utilizam o banco de dados e seus objetos, direta ou indiretamente e estão agrupados nas seguintes categorias:

- **desenvolvedor de aplicações:** é o profissional responsável pela construção de aplicações que irão acessar a base de dados para realizar consultas, alterações ou exclusões de dados;
- **usuário final:** É a pessoa que faz uso dos dados consolidados para tomada de decisão ou simples consulta ou atualização. Normalmente interage com o banco por intermédio de aplicações.
- **analista de Banco de dados (Case):** É o usuário responsável pela modelagem dos dados e implementação do banco de dados nos níveis lógico e físico junto com o DBA.
- **administrador de banco de dados:** É o profissional que tem as seguintes responsabilidades:

    - Instalar e atualizar o banco de dados e as ferramentas de aplicação;

---

[6] Este tópico será estudado na parte 2.

- Alojar sistemas de armazenamento e planejar os futuros armazenamentos de requerimentos para o sistema de banco de dados;
- Criação e armazenamento das estruturas primárias para que os desenvolvedores possam gerar aplicações;
- Criação de objetos primários, uma vez que os usuários tenham construído uma nova aplicação;
- Modificar a estrutura do banco de dados para adequá-lo às novas aplicações;
- Garantir a disponibilidade do banco de dados, bem como a performance do mesmo;
- Controlar e monitorar os acessos dos usuários ao banco de dados (Segurança);
- Fazer o backup das informações e a restauração;
- Manter um controle sobre os backup's.

**ATENÇÃO:** Os usuários têm acesso ao banco de dados ou aos seus objetos de acordo com os privilégios que lhe são conferidos.

## 2. SQL – Strutured Query Language

Neste capítulo serão abordadas as estruturas SQL para criação e manipulação de tabelas e dados. A SQL é uma linguagem estruturada que utiliza uma combinação de construtores em álgebra e cálculo relacional e que possibilita não só a realização de consultas, como o próprio nome sugere, mas também a manipulação de tabelas e dados. apesar de ser conhecida como uma linguagem para consultas possibilita a realização de consultas, mas também criação, alteração e exclusão de tabelas e manipulações de dados. A SQL é a linguagem padrão utilizada pelos SGBD relacionais, mesmo assim podem existir algumas variações quanto à utilização. Neste curso a SQL será abordada de acordo com as especificidades do SGBD relacional Oracle.

### 2.1. Histórico

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM, desenvolveu a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, início dos anos 70, originalmente chamada de SEQUEL - Structured English Query Language.
Em 1977 foi revisada e passou a ser chamada de SQL - Structured Query Language.
Em 1986 o American National Standard Institute (ANSI), publicou um padrão SQL, então a SQL estabeleceu-se como linguagem padrão para Bancos de Dados Relacionais.

### 2.2. Características da SQL

A SQL possui comandos para a definição dos dados (DDL-Data Definition Language), comandos para a manipulação de dados (DML – Data Manipulation Language) e uma subclasse de comandos DML, a DCL (Data Control Language), dispõe de comandos de controle como Grant e Revoke.

Algumas características da SQL são:

- **Independência de fabricante:** A SQL é oferecida em praticamente todos os SGBDs.
- **Portabilidade entre computadores:** A SQL pode ser utilizada desde um PC, passando por Workstantions, até Mainframes.
- **Redução de custos com treinamento:** Devido à portabilidade, as aplicações podem se movimentar de um ambiente para outro sem necessidade de um novo treinamento.

- **Inglês estruturado de alto nível:** A SQL é formada por um conjunto bem simples de sentenças em inglês, oferecendo um rápido e fácil entendimento.
- **Consulta interativa:** Provê um acesso rápido aos dados, fornecendo respostas ao usuário quase instantaneamente.
- **Múltiplas visões dos dados:** Permite ao criador do banco de dados levar diferentes visões dos dados a diferentes usuários.
- **Definição dinâmica dos dados:** Pode-se alterar, expandir ou incluir, dinamicamente, as estruturas dos dados armazenados.

## 2.3. Instruções DDL

Os comandos DDL (Data Definition Language) composta entre outros pelos comandos Create, que é destinado a criação do Banco de Dados, das Tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop. Por meio das instruções DDL podem ser realizadas as tarefas de:

- Criação, alteração e eliminação de objetos do banco de dados, como por exemplo tabelas, índices, seqüências;
- Conceção e revogação de privilégios em objetos
- Criação do banco de dados
- Criação de usuários;
- Conceção e revogação de privilégios à usuários

## 2.4. Instruções DML

Os comandos da série DML (Data Manipulation Language), destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML pode-se citar os comandos:

- Insert – Inserção de dados
- Update – Alteração de dados
- Delete – Remoção de dados
- Commit – Confirmação das manipulações
- Rollback – Desistência das manipulações
- Select [7]– Seleciona linhas de dados de tabelas ou visões

## 2.5. Instruções DCL

Os comandos da série DCL (Data Control Language) são utilizados para controlar os privilégios de usuários, com eles é possível:

- permitir a um usuário que se conect ao banco;

[7] Select - Existem autores que a classificam com instrução do DRL – Linguagem para Recuperação de Dados.

- permitr que crie objetos no banco;
- permitir que consulte objetos do banco; entre outros
- cancelar os privilégios de um usuário ou personagem.

Para isso, estão disponíveis as instruções:

Grant: utilizada para conceder privilégios aos usuários
Revoke: Utilizada para cancelar os privilégios dos usuários

### 2.6. Ambiente de trabalho

Será utilizada como ferramenta para edição e execução de instruções SQL o SQL*Plus.

O SQL*PLUS é uma ferramenta da Oracle usada como interface para acesso ao banco de dados Oracle, não é gráfica[8] e é capaz de reconhecer e executar instruções SQL e PL/SQL[9]. As instruções podem ser escritas diretamente no prompt ou em um editor de textos.
O SQL*Plus pode ser acessado a partir de um duplo clique no ícone do aplicativo .ou selecionando diretamente o arquivo que está localizado no diretório c:**\oracle\ora81\bin\sqlplusw.exe**
A seguir será solicitada a identificação do usuário e nome do banco que será utilizado. Alguns usuários e senhas default são:

Figura 1 - ìcone do SQLl*Plus

| Usuário | Senha | Privilégios |
|---|---|---|
| Scott | Tiger | usuário |
| Internal | Oracle | administrador |

Após a autenticação da conexão será aberto o ambiente do SQL*Plus. A ferramenta oferece um menu com opções para manipulação de arquivos, configuração do ambiente e outras.
As instruções são digitadas no prompt e submetidas à execução após o enter. Também podem ser executados

Figura 2 - Autenticação e conexão

[8] Como ferramenta gráfica poderá ser utilizado, por exemplo, o ORACLE NAVIGATOR: é uma ferramenta gráfica do Oracle que permite ao desenvolvedor a criação e manutenção de objetos no banco de dados.
[9] PL/SQL: É a linguagem procedural do SQL do ORACLE, composta essencialmente de todos os comandos SQL padrão e outras instruções tais como estruturas de seleção, estruturas de repetição, recursos de manipulação de cursores, outras que permitem utilizar o SQL de forma procedural. Será estudada com mais detalhes na parte 3 da disciplina.

arquivos com extensão sql, para isso deve-se digitar o comando execute seguido do caminho e nome do arquivo.
Todos os exemplos apresentados neste material e exercícios serão trabalhados, a príncipio neste ambiente.
Para a edição de instruções no SQL*PLUS deve-se considerar algumas recomendações:

- Não é feita distinção entre maiúsculas de minúsculas, a menos que indicado.
- As instruções SQL podem ser digitadas em uma ou mais linhas.
- As palavras-chave não podem ser divididas entre as linhas nem abreviadas.
- As cláusulas são em geral colocadas em linhas separadas para melhor legibilidade e facilidade de edição.
- As guias e endentações podem ser usadas para tornar o código mais legível. Em geral, as palavras-chave são digitadas em letras maiúsculas, todas as outras palavras, como nomes de tabela e colunas são digitadas em minúsculas.
- Dentro do SQL*Plus, uma instrução SQL é digitada no prompt SQL e as linhas subseqüentes são numeradas. Isso chama-se *buffer de SQL*. Somente uma instrução pode ser a atual a qualquer momento dentro do buffer.

## 2.7. Criando Tabelas

Está é uma operação DDL. Para criar uma tabela é necessário que o usuário tenha privilégio e uma área para armazenamento. A sintaxe simplificada para criação de tabelas é:

**Sintaxe:**
```
CREATE TABLE [esquema.]tabela
    (nome da coluna tipo do dado [DEFAULT expr]
            [constraint da coluna],
            ...,
            [constraint da tabela]);
```

onde:

| | |
|---|---|
| *Esquema:* | é o nome do proprietário da tabela, quando omitido a tabela é criada no esquema do usuário corrente |
| *Tabela:* | é o nome da tabela |
| *DEFAULT expr:* | especifica um valor default que será utilizado quando um dado for omitido na inserção |
| *Coluna:* | é o nome da coluna |
| *tipo de dados:* | é o tipo de dados e o comprimento da coluna |
| *Constraint:* | Está cláusula é opcional e especifica as restrições para a coluna ou para a tabela, quando o nome da constraint é omitido o Oracle assume uma identificação |

Convenções para Nomeação de Tabelas e Colunas:

- Deve começar com uma letra
- Pode ter de 1 a 30 caracteres
- Deve conter somente A–Z, a–z, 0–9, _, $ e #
- Não deve duplicar o nome de outro objeto de propriedade do mesmo usuário
- Não deve ser uma palavra reservada pelo Oracle Server

Tipos de Dados:

| **Tipo de Dados** | **Descrição** |
|---|---|
| `VARCHAR2(`*tamanho*`)` | Dados de caractere de comprimento variável |
| `CHAR(`*tamanho*`)` | Dados de caractere de comprimento fixo |
| `NUMBER(`*p*`,`*s*`)` | Dados numéricos de comprimento variável |
| `DATE` | Valores de data e hora |
| `LONG` | Dados de caractere de comprimento variável até 2 gigabytes |
| `CLOB` | Dados de caractere de um byte de até 4 gigabytes |
| `RAW e LONG RAW` | Dados binários brutos |
| `BLOB` | Dados binários de até 4 gigabytes |
| `BFILE` | Dados binários armazenados em um arquivo externo de até 4 gigabytes |

No exemplo abaixo está sendo criada a tabela DEPT, com três colunas — chamadas, DEPTNO, DNAME e LOC.

```
SQL>CREATE TABLE dept
  2  (deptno NUMBER(2),
  3  dname      VARCHAR2(14),
  4  loc   VARCHAR2(13));
Table created.
```

A instrução Describe é utilizada para exibir a estrutura de uma tabela.

```
SQL> DESCRIBE dept

Name                    Null?    Type
------------------- -------- ---------
DEPTNO                           NUMBER(2)
DNAME                            VARCHAR2(14)
LOC                              VARCHAR2(13)
```

### 2.7.1. Restrições (constraints)

As restrições impõem regras que podem ser no nível da coluna ou no nível da tabela. São utilizadas para impedir que dados inválidos sejam digitados nas tabelas, garantindo assim a consistência dos dados.
Os seguintes tipos de restrição são válidos no Oracle:

- **NOT NULL –** Impõe a inserção obrigatória de dados nas colunas com está restrição;
- **UNIQUE** – Campos com está restrição não aceitam dados com valores já inseridos em outros registros
- **PRIMARY KEY –** Define uma ou mais colunas como chave primária da tabela.
- **FOREIGN KEY –** Define uma ou mais colunas como chave estrangeira da tabela.
- **CHECK –** Especifica uma lista de valores que serão utilizados para validar a inserção de um dado

Todas as restrições são armazenadas no dicionário de dados, as restrições não nomeadas recebem serão identificadas pelo oracle com o formato SYS_cn, onde *n* é um número inteiro para criar um nome de restrição exclusivo.
As restrições podem ser definidas enquanto a tabela está sendo criada ou adicionadas após sua criação, podendo ainda, ser desativadas temporariamente.

#### 2.7.1.1 Restrição NOT NULL

A restrição NOT NULL assegura que os valores nulos não sejam permitidos na coluna. As colunas sem uma restrição NOT NULL podem conter valores nulos por default. Deve ser definida no nível da coluna

**Exemplo:** No exemplo acima a restrição NOT NULL está sendo aplicada às colunas ENAME e DEPTNO da tabela EMP. Observe que na linha 3 a restrição está sendo identificada, já na linha 9 não, neste caso o Oracle a identificará de acordo com o seu padrão de identificação *SYS_cn*.

```
SQL> CREATE TABLE emp(
  2   empno      NUMBER(4),
  3   ename      VARCHAR2(10) constraint emp_ename_NN NOT NULL,
  4   job   VARCHAR2(9),
  5   mgr   NUMBER(4),
  6   hiredate   DATE,
  7   sal   NUMBER(7,2),
  8   comm  NUMBER(7,2),
  9   deptno     NUMBER(7,2) NOT NULL);
```

Para verificar se as colunas da tabela estão ou não com a restrição NOT NULL, utilize a instrução DESCRIBE, como segue exemplo abaixo:

```
SQL> DESCRIBE EMP;

Name                          Null?        Type
EMPNO                        NOT NULL      NUMBER(4)
ENAME                                      VARCHAR2(10)
JOB                                        VARCHAR2(9)
MGR                                        NUMBER(4)
HIREDATE                                   DATE
SAL                                        NUMBER(7,2)
COMM                                       NUMBER(7,2)
DEPTNO                       NOT NULL      NUMBER(2)
```

#### 2.7.1.2 Restrição UNIQUE KEY

Uma restrição de integridade UNIQUE KEY requer que cada valor em uma coluna ou conjunto de colunas (chave) seja exclusivo — ou seja, duas linhas de uma tabela **não podem ter valores duplicados** em uma coluna específica ou conjunto de colunas. A coluna (ou conjunto de colunas) incluída na definição da restrição UNIQUE KEY é chamada de *chave exclusiva*. Se a chave UNIQUE contiver mais de uma coluna, tal grupo de colunas é considerado uma *chave exclusiva composta*. Podem ser definidas no nível da coluna ou da tabela

**Exemplo:** Criar a tabela dept cujo nome do departamento (dname) não poderá ser duplicado.

```
SQL> CREATE TABLE   dept(
  2   deptno     NUMBER(2),
  3   dname VARCHAR2(14),
  4   loc   VARCHAR2(13),
  5   CONSTRAINT dept_dname_uk UNIQUE(dname));
```

No exemplo anterior a constraint dept_dname_uk está sendo criada no nível da tabela

#### 2.7.1.3 Restrição PRIMARY KEY

Uma restrição PRIMARY KEY cria uma única chave primária para cada tabela. A restrição PRIMARY KEY é uma coluna ou conjunto de colunas que identifica exclusivamente cada linha em uma tabela. Essa restrição impõe a exclusividade da coluna ou combinação de colunas e assegura que nenhuma coluna que seja parte da chave primária possa conter um valor nulo. Pode ser definida no nível da tabela ou da coluna

**ATENÇÃO:** Um índice UNIQUE é automaticamente criado para uma coluna PRIMARY KEY.

**Exemplo:** Criar a tabela dept cujo campo número do departamento (deptno) deverá ser a chave de identificação do registro:

**Resolução 1 –** Definição da chave primária no nível da tabela:

```
SQL> CREATE TABLE   dept(
  2  deptno     NUMBER(2),
  3  dname      VARCHAR2(14),
  4  loc        VARCHAR2(13),
  5  CONSTRAINT dept_dname_uk UNIQUE (dname),
  6  CONSTRAINT dept_deptno_pk PRIMARY KEY(deptno));
```

**Resolução 2 –** Definição da chave primária no nível da coluna:

```
SQL> CREATE TABLE   dept(
  2  deptno NUMBER(2)CONSTRAINT dept_deptno_pk PRIMARY KEY,
  3  dname      VARCHAR2(14),
  4  loc        VARCHAR2(13),
  5  CONSTRAINT dept_dname_uk UNIQUE (dname);
```

**ATENÇÃO:** A definição de uma chave primária composta deve ser feita no nível da tabela:

```
SQL> CREATE TABLE   pk_composta(
  2  valor1 NUMBER(2),
  3  valor2 NUMBER(2),
  4  CONSTRAINT dept_deptno_pk PRIMARY KEY(valor1, valor2));
```

#### 2.7.1.4 Restrição FOREIGN KEY

É uma restrição de integridade referencial, designa uma coluna ou combinação de colunas como a chave estrangeira e estabelece um relacionamento entre a chave primária ou uma chave exclusiva na mesma tabela ou em uma tabela diferente. Um valor de chave estrangeira deve corresponder a um valor existente na tabela mãe ou ser NULL.
As chaves estrangeiras são baseadas nos valores dos dados, sendo puramente lógicas, e não ponteiros físicos. Pode ser definida no nível da tabela ou da coluna.

**Exemplo:** No exemplo a seguir o DEPTNO foi definida como a chave estrangeira na tabela EMP (tabela filha ou dependente); ela faz referência à coluna DEPTNO da tabela DEPT (tabela mãe ou referenciada).

**Resolução 1** – Restrição Foreign Key definida no nível da tabela

```
SQL> CREATE TABLE    emp(
  2  empno      NUMBER(4),
  3  ename VARCHAR2(10) NOT NULL,
  4  job   VARCHAR2(9),
```

```
 5  mgr  NUMBER(4),
 6  hiredate  DATE,
 7  sal  NUMBER(7,2),
 8  comm NUMBER(7,2),
 9  deptno    NUMBER(7,2) NOT NULL,
10  CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
11            REFERENCES dept (deptno) ON DELETE CASCADE );
```

| **TABELA EMP** | **TABELA DEPT** |
|---|---|
| empno **pk** | deptno **pk** |
| ename | dname |
| job | loc |
| mgr | |
| hiredate | |
| sal | |
| deptno **fk** | |
| comm | |

Observe que o campo deptno é chave primária na tabela DEPT e está sendo utilizado na tabela EMP como chave estrangeira. Por meio destas duas colunas será possível relacionar as tabelas EMP e DEPT.

**Resolução 2** – Restrição Foreign Key definida no nível da coluna, observe que a instrução foreign key não é requerida.

```
SQL> CREATE TABLE  emp(
  2      empno    NUMBER(4),
  3  ename    VARCHAR2(10) NOT NULL,
  4  job  VARCHAR2(9),
  5  mgr  NUMBER(4),
  6  hiredate  DATE,
  7  sal  NUMBER(7,2),
  8  comm NUMBER(7,2),
  9 deptno NUMBER(7,2) constraint  emp_deptno_fk
 10                    REFERENCES dept (deptno) not null));
```

**ATENÇÃO:** A definição de uma chave estrangeira composta deve ser feita no nível da tabela:

### 2.7.1.5 Restrição CHECK

Define uma condição que cada registro deve atender. Podem ser utilizados operadores de comparação para delimitação dos valores a serem aceitos para a coluna.

**Exemplo:** Para evitar erros de digitação do usuário você coloca uma restrição do tipo CHECK para o campo SALÁRIO, onde o mesmo deve ser maior que o salário mínimo de R$200,00, desta forma ao tentar inserir valor menores que 200 a restrição será ativada e será apresentado um erro para o usuário.

**Resolução 1** – Restrição criada no nível da tabela:

```
SQL> CREATE TABLE  emp(
2         empno  NUMBER(4),
3         ename VARCHAR2(10) NOT NULL,
4         job VARCHAR2(9),
5         mgr NUMBER(4),
6         hiredate DATE,
7         sal NUMBER(7,2),
8         constraint emp_sal_ck check (sal > 200));
```

**Resolução 2** – Restrição criada no nível da  coluna:

```
SQL> CREATE TABLE  emp(
2               empno  NUMBER(4),
3               ename VARCHAR2(10) NOT NULL,
4               job VARCHAR2(9),
5               mgr NUMBER(4),
6               hiredate DATE,
7               sal NUMBER(7,2) check (sal > 200));
```

### 2.7.1.6 Consultando restrições de uma tabela

A instrução SELECT, que será estudada com mais detalhes em um tópico posterior, é utilizada para realização de consultas.

**Sintaxe:**

```
 Select nome_coluna1, nome_coluna2 ... nome_colunaN
from nome_tabela1, nome_tabelaN
where condição;
```

No exemplo a seguir serão consultadas as restrições definidas para a tabela emp:

```
SQL>SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE,
2   STATUS,SEARCH_CONDITION
3   FROM USER_CONSTRAINTS
4   WHERE TABLE_NAME = 'EMP';
```

onde:

`CONSTRAINT_NAME:` é o nome da restrição

`CONSTRAINT_TYPE:` tipos de restrições aos quais os campos estão envolvidos, onde:
C = NOT NULL
P = PRIMARY KEY
R = FOREIGN KEY e
U = UNIQUE KEY

`STATUS:` representa o estado em que a restrição se encontra: ENABLE, significa que a restrição está válida e esta sendo usada e DISABLE que a restrição esta desabilitada e que por isso não está em uso.

`SEARCH_CONDITION:` é a condição expressa da restrição.

**Resultado:**

```
CONSTRAINT_NAME      C     STATUS     SEARCH_CONDITION
------------------    -    --------   --------------------
SYS_C00884           C     ENABLED    "EMPNO" IS NOT NULL
SYS_C00885           C     ENABLED    "DEPTNO" IS NOT NULL
EMP_EMPNO_PK         P     ENABLED
EMP_MGR_FK           R     ENABLED
EMP_DEPTNO_FK        R     ENABLED
EMP_ENAME_UK         U     ENABLED
```

## 2.8. Inserção de Dados

A inserção de dados é uma operação DML.

**Sintaxe:**

**INSERT INTO** *tabela* [(*coluna* [, *coluna...*])] **VALUES** *(valor* [, *valor...*]);

Onde:

| | |
|---|---|
| Tabela | é o nome da tabela |
| Coluna | é o nome da coluna a ser preenchida, a lista de colunas pode ser omitida, neste caso devem ser informados valores para todas as colunas. |
| Valor | é o valor correspondente para a coluna. Os valores de data e caractere devem ser informados entre aspas simples. |

Para verificar a ordem default das colunas de uma tabela e o tipo de dado esperado utilize a instrução describe.

```
SQL> DESCRIBE  dept
```

| **Name** | **Null?** | **Type** |
|---|---|---|
| DEPTNO | NUMBER(2) | NOT NULL |
| DNAME | VARCHAR2(14) | |
| LOC | VARCHAR2(13) | |

**Exemplo 1:** Instrução completa para inserção de dados

```
SQL> INSERT INTO    dept (DEPTNO, DNAME, LOC)
  2  VALUES         (70, 'PRODUCAO', ‘MARILIA’);
```

Neste caso, não existe a necessidade de indicar as colunas que irão receber as inserções, pois estão sendo inseridos valores para todas as colunas, então o mesmo exemplo poderia ser resolvido assim:

**Exemplo 2:**
```
SQL> INSERT INTO    dept
  2  VALUES         (70, 'PRODUCAO', ‘MARILIA’);
```

Também é possível  a inserção implícita de nulos, isto é, pode-se deixar de informar uma coluna na inserção de dados, para a qual será atribuído nulo:

**Exemplo 3:**
```
SQL> INSERT INTO    dept (deptno, dname )
  2  VALUES         (60, 'MIS');
```

A tabela dept contém as coluna deptno, dname e loc, no entanto no exemplo anterior estão sendo inseridos valores apenas às colunas deptno e dname, neste caso a coluna loc irá conter NULL.

**Exemplo 4:**
```
SQL> INSERT INTO    dept
  2  VALUES         (70, 'FINANCE', NULL);
```

Já neste outro exemplo o nulo para a coluna loc está sendo explicitado, note que a lista de colunas é omitida pois estão sendo mencionados os dados para todas as colunas. A palavra NULL poderia ser substituida por aspas simples (70, 'FINANCE', '').

Para inserção de datas o formato default do Oracle é DD-MON-YY, mas a data também pode ser informada de acordo com a configuração do sistema, veja o exemplo a seguir:

**Exemplo 5:**
```
SQL> INSERT INTO emp
  2  VALUES      (2296,'AROMANO','SALESMAN',7782, '03/02/97',
  3   1300, NULL, 10);
```

O resultado pode ser visualizado executando a consulta:

```
SQL> Select *
2    from emp;
```
[10]

```
EMPNO  ENAME    JOB       MGR  HIREDATE   SAL   COMM   DEPTNO
-----  -------  --------  ---- --------   ----  ----   ------
2296   AROMANO  SALESMAN  7782 03/02/97   300          10
```

[10] A instrução select será abordada com mais detalhes posterioremente.

## 2.9. Alteração de dados

A alteração de dados é uma operação DML.

**Sintaxe:**

```
UPDATE tabela
SET coluna = valor [, coluna = valor, ...]
[WHERE condição];
```

onde:

| | |
|---|---|
| Tabela | é o nome da tabela |
| Coluna | é o nome da coluna a ser preenchida, podem ser atualizados os dados de várias colunas |
| Valor | é o valor correspondente ou subconsulta para a coluna |
| Condição | é uma cláusula opcional e identifica as linhas a serem atualizadas de acordo com uma condição que pode ser composta por expressões de comparação ou subconsultas |

**Exemplo:** Alterar o códido do departamento para 20 do funcionário com código 7782:

```
SQL> UPDATE    emp
  2  SET       deptno = 20
  3  WHERE     empno = 7782;
```

**ATENÇÃO:** Todas as linhas na tabela são modificadas quando a cláusula WHERE é omitida.

## 2.10. Remoção de dados

A remoção de dados é uma operação DML.

**Sintaxe:**

```
DELETE [FROM]tabela [WHERE condição];
```

onde:

| | |
|---|---|
| Tabela: | é o nome da tabela |
| Condição: | Está cláusula é opcional e identifica as linhas a serem eliminadas de acordo com uma condição que pode ser composta por expressões de comparação ou subconsultas |

**Exemplo:**

```
SQL> DELETE FROM dept
  2  WHERE dname = 'PRODUCAO';
```

**ATENÇÃO:** Todas as linhas na tabela serão removidas se a cláusula WHERE for omitida.

### 2.11. Confirmando ou Descartando transações

Do ponto de vista do usuário uma transação parece uma simples operação, por exemplo transferir o dinheiro de uma conta corrente para outra o usuário informa os dados requeridos para que a operação de transferência seja realizada e recebe uma notificação de conclusão da operação. Por outro lado, para que a operação seja realizada com sucesso, sem a ocorrência de falhas, uma série de operações devem ser realizadas, no caso do exemplo de transferência de valores de uma conta para outra de maneira bem simplificada:

- o dinheiro precisa ser debitado de uma conta, portanto uma operação de atualização de dados deve ser realizada nesta conta;
- o dinheiro precisa ser creditado em outra conta, portanto outra operação de atualização de dados deve ser realizada nesta outra conta.

Sendo assim, uma transação é um conjunto de operações DML que são realizadas para concluir uma determinada tarefa.
Para garantir a integridade dos dados é necessário que as transações assegurem:

- A consistência de dados
- A atomicidade – todas as operações devem ser refletidas corretamente no banco ou então nenhuma das operações deverá ser realizada
- A integridade da base de dados

Quando não ocorrem falhas no processamento das operações de uma transação ela pode ser efetivada, neste caso as modificações são refletidas fisicamente no banco de dados. Enquanto isso não ocorre, as modificações são refletidas apenas em memória e podem ser desfeitas ou descartadas.
O controle de transações também permite que as alterações realizadas, possam ser visualizadas antes de se tornarem permanentes, e que as operações relacionadas logicamente possam ser agrupadas.
As instruções responsáveis pelo controle das transações são o commit e o rollbback. A emissão de um commit confirma as transações, isto é, efetiva as manipulações de dados realizadas nas tabelas. A emissão de um rollback descarta as transações que ainda não tenham sido confirmadas.
Uma transação tem início quando a primeira instrução SQL executável é realizada e termina com um dos seguintes eventos:

- a emissão de uma instrução COMMIT ou ROLLBACK;
- a execução de uma instrução DDL ou DCL, nesse caso ocorre um commit automático;
- quando o usuário sai do SQL*Plus; ou
- Houver uma falha no computador ou o sistema cair.
- Um commit automático ocorre sob as seguintes circunstâncias:
- A instrução DDL é emitida
- A instrução DCL é emitida
- A saída normal do SQL*Plus, sem emitir explicitamente COMMIT ou ROLLBACK

Um ROLLBACK automático ocorre quando há uma finalização anormal do SQL*Plus ou queda do sistema.
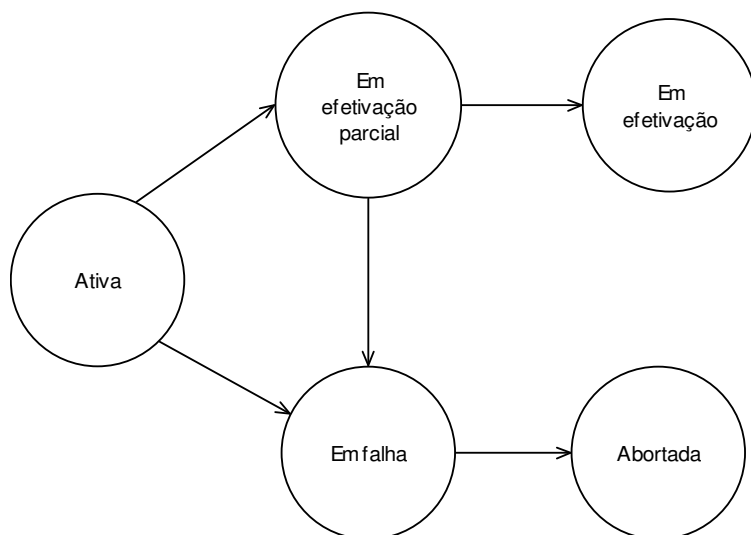
Figura 3 - Diagrama de estados de uma transação - fonte: Silberschatz [1999]

Uma transação pode estar:

- ativa: enquanto suas operações estão sendo executadas;
- em efetivação parcial: após a execução da última operação, antes do commit
- em falha: quando alguma das operações não pode ser realizada normalmente
- abortada: quando a transação é desfeita (rollback) e o banco de dados volta ao estado anterior ao início da transação
- em efetivação: quando concluída com sucesso (commit).

**Exemplo usando commit:** Atualizar a tabela EMP e definir o número de departamento para o funcionário 7782 (Clark) como 10 e depois confirmar a alteração.

| | |
|---|---|
| `SQL> UPDATE emp`<br>`2 SET deptno = 10` | Início da transação; transação ativa; uma cópia dos dados antes da alteração, é armazenada no segmento de rollback |
| `3 WHERE empno = 7782;` | Em efetivação parcial; conclusão da operação; A alteração está disponível em memória; as linhas afetadas ficam bloqueadas; |
| **`SQL> COMMIT;`** | Em efetivação; a alteração é refletida fisicamente no banco; As linhas afetadas são desbloqueadas.<br>A imagem antiga é retirada do segmento de rollback.[11] |

[11] Segmento de rollback – arquivo que armazena uma cópia dos dados de uma tabela que estão envolvidos em transações.

O bloqueio implícito ocorre para todas as instruções SQL exceto SELECT. O mecanismo de bloqueio default do Oracle automaticamente usa o nível mais inferior da restrição aplicável, fornecendo assim o maior grau de simultaneidade e máxima integridade de dados. O bloqueio em um banco de dados do Oracle é automático e não requer ação do usuário.

**ATENÇÃO:** É importante lembrar que após a emissão do comando COMMIT não há mais como reverter a operação feita anteriormente.

**Exemplo usando rollback:** Remover todas as linhas da tabela employee

| | |
|---|---|
| `SQL> Delete` | Início da transação; transação ativa; uma cópia dos dados antes da alteração, é armazenada no segmento de rollback |
| `  2  From employee;` | Em efetivação parcial; conclusão da operação; A alteração está disponível em memória; as linhas afetadas ficam bloqueadas |
| `  14 rows deleted` | |
| **`SQL> ROLLBACK;`** | Transação abortada; as modificações são desfeitas; As linhas afetadas são desbloqueadas. A imagem antiga é retirada do segmento de rollback. |

A versão original, mais antiga, dos dados no segmento de rollback é gravada de volta na tabela.
Por meio da consistência na leitura dos dados que cada usuário visualiza os dados como eles eram no último commit, antes da operação DML iniciar, com isso assegura que:
Os usuários não vejam os dados que estejam sendo alterados.
As alterações feitas por um usuário não interrompam nem entram em conflito com as alterações que outro usuário esteja fazendo.
Linhas que estão sendo alteradas são disponibilizadas apenas para consulta.
A implementação da consistência de leitura é automática, mantém uma cópia parcial do banco de dados em segmentos de rollback, isto é, quando uma operação de inserção, atualização ou exclusão é feita no banco de dados, o Oracle Server tira uma cópia dos dados antes de serem alterados e os grava no segmento de rollback.

Para descobrir o nome da restrição pode-se consultar as views[15] do dicionário de dados USER_CONSTRAINTS ou USER_CONS_COLUMNS. Antes de executar a consulta verifique os nomes das colunas com a instrução DESCRIBE.

**Exemplo:**

```
SELECT CONSTRAINT_NAME, COLUMN_NAME
FROM  USER_CONS_COLUMNS
WHERE  TABLE_NAME = 'EMP';
```

**Resultado:**

```
CONSTRAINT_NAME        COLUMN_NAME
EMP_DEPTNO_FK             DEPTNO
EMP_EMPNO_PK              EMPNO
EMP_ENAME_UK              ENAME
EMP_MGR_FK               MGR
SYS_C00884                EMPNO
SYS_C00885                DEPTNO
```

#### 2.14.1.5 Desativando e Ativando uma Restrição

Uma restrição pode ser desativada com a utilização da cláusula DISABLE em conjunto com a instrução ALTER TABLE. Para torna-la ativa novamente utiliza-se a cláusula ENABLE.

**Sintaxe:**

```
ALTER TABLE   tabela
DISABLE/ENABLE  CONSTRAINT restrição [CASCADE];
```

onde:

| | |
|---|---|
| *Tabela* | é o nome da tabela |
| *Restrição* | Nome da constraint que será eliminada |
| *Cascade* | Está cláusula elimina todas as restrições dependentes |

**Exemplo 1:** Desativar a chave primária da tabela emp

```
SQL> ALTER TABLE      emp
  2  DISABLE CONSTRAINT    emp_empno_pk CASCADE;
```

**Exemplo 2:** Ativar a chave primária da tabela emp

```
SQL> ALTER TABLE      emp
  2  ENABLE CONSTRAINT     emp_empno_pk;
```

Pode-se usar a cláusula DISABLE nas instruções CREATE TABLE e ALTER TABLE.

---

[15] As views do dicionário de dados são “tabelas” lógicas que contém informações sobre os objetos do banco de dados.

**ATENÇÃO:** Para tornar uma restrição ativa os dados inseridos na coluna devem estar em conformidade com a restrição, isto é devem respeita, caso contrário não será possível.

### 2.14.2. Eliminando uma Tabela

A instrução DROP TABLE remove a definição de uma tabela. Quando uma tabela é eliminada, o banco de dados perde todos os dados na tabela e todos os índices associados a ela.

Sintaxe:

```
DROP TABLE tabela;
```

onde:

*Tabela* é o nome da tabela

**Exemplo:** Eliminar a tabela dept30.

```
SQL> DROP TABLE dept30;
```

Ao eliminar uma tabela:

- Todos os dados são deletados da tabela.
- As views e sinônimos permanecerão, mas serão inválidos.
- Todas as transações pendentes sofrerão commit.
- Somente o criador da tabela ou um usuário com o privilégio DROP ANY TABLE poderá remover uma tabela.

**ATENÇÃO:** A instrução DROP TABLE, uma vez executada, é irreversível. O Oracle Server não questiona a ação quando você emite a instrução DROP TABLE. Se você possuir tal tabela ou tiver um privilégio de nível superior, então a tabela será imediatamente removida.

## 2.15. Visões (Views)

Uma view é uma tabela lógica baseada em uma tabela ou outra view. Uma view não contém dados próprios mas é como uma janela através da qual os dados das tabelas podem ser vistos ou alterados. As tabelas nas quais uma view é baseada são chamadas *tabelas-base*. A view é armazenada como uma instrução SELECT no dicionário de dados.
Podem ser utilizadas para:

- Restringir o acesso a dados
- Facilitar as consultas complexas, por exemplo, possibilita que usuários consultem informações de várias tabelas sem saber como criar uma instrução de junção.
- Permitir a independência dos dados para usuários ad hoc e programas aplicativos.
- Apresentar diferentes visões dos mesmos dados

### 2.15.1. Criando Visões

**Sintaxe:**

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nome_view
  [(apelido[, apelido]...)]
AS subconsulta
[WITH CHECK OPTION [CONSTRAINT restrição]]
[WITH READ ONLY];
```

onde:

| | |
|---|---|
| CREATE VIEW  OR REPLACE | Cria uma view ou substitui a view existente |
| nome_view | Identificador da view |
| FORCE | Cria a view independentemente das tabelas base existirem ou não |
| NOFORCE | Cria a view somente se as tabelas base existirem (default) |
| *apelido* | especifica apelidos para as colunas selecionadas pela view, é opcional, mas se utilizados devem corresponder ao número de colunas selecionadas na consulta. |
| consulta da view | Operação de consulta às tabelas base ou a outras views para gerar o resultado da visão. |
| *subconsulta* | é uma instrução SELECT completa (Você pode usar apelidos para as colunas na lista SELECT.) e não pode conter a cláusula order by |
| WITH CHECK OPTION | especifica que somente linhas acessíveis à view podem ser inseridas ou atualizadas |
| *restrição* | é o nome atribuído à restrição CHECK OPTION |
| WITH READ ONLY | assegura que as operações DML não possam ser executadas nesta view |

Há duas classificações para views: simples e complexa:

Uma ***view simples*** é uma que:

- Seleciona dados a partir de somente uma tabela
- Não contém funções ou grupos de dados
- Pode-se executar a DML

Uma ***view complexa*** é uma que:

- Seleciona dados a partir de várias tabelas
- Contém funções ou grupos de dados
- Nem sempre pode-se executa a DML

**Exemplo 1:** Criar uma view com nome EMPVU10, que contenha detalhes tais como número, nome e cargo dos funcionários que trabalham no departamento 10.

```
SQL> CREATE VIEW      empvu10
  2  AS SELECT  empno, ename, job
  3  FROM             emp
  4  WHERE            deptno = 10;
```

A estrutura da view pode ser exibida utilizando o comando DESCRIBE do SQL*Plus:

```
SQL> DESCRIBE EMPVU10;

Name                            Null?         Type
EMPNO                         NOT NULL        NUMBER(4)
ENAME                                         VARCHAR2(10)
JOB                                           VARCHAR2(9)
```

**ATENÇÃO:** Se não for especificado um nome de restrição para uma view criada com CHECK OPTION, o sistema irá atribuir um nome default no formato SYS_C*n*.
A opção OR REPLACE altera a definição da view sem eliminá-la, recriá-la ou reconceder-lhe os privilégios de objetos anteriormente concedidos.

**Exemplo 2:** Alterar a definição da view empvu10 para que sejam exibidos apelidos para as colunas da consulta: id_emp, nome e cargo:

```
SQL> CREATE OR REPLACE VIEW empvu10
  2        (id_emp, nome, cargo)
  3  AS SELECT  empno, ename, job
  4  FROM                   emp
  5  WHERE                  deptno = 10;
```

**Exemplo 3:** Criar uma view para exibir o nome do departamento e o menor salário, maior salário e a média salarial de cada departamento.

```
SQL> CREATE VIEW    dept_sum_vu
  2                        (name, minsal, maxsal, avgsal)
  3  AS SELECT d.dname, MIN(e.sal), MAX(e.sal),
  4                 AVG(e.sal)
  5  FROM                  emp e, dept d
  6  WHERE                       e.deptno = d.deptno
  7  GROUP BY  d.dname;
```

No exemplo anterior está sendo criada uma view complexa para exibir os nomes de departamento, maior salário, menor salário e o salário médio por departamento. Note que os nomes alternativos foram especificados para a view. Esse é um requisito se uma coluna da view derivar de uma função ou expressão.

**Exemplo 4:** Criar uma view, identificada por empvu20, com todas as colunas da tabela emp e todos os funcionário do departamento 20. Acrescentar uma cláusula para garantir que as manipulações permaneçam no domínio da view.

```
SQL> CREATE OR REPLACE VIEW empvu20
  2  AS SELECT  *
  3  FROM          emp
  4  WHERE         deptno = 20
  5  WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

No exemplo anterior qualquer tentativa de alteração do número do departamento para qualquer linha na view falhará porque ela violará a restrição WITH CHECK OPTION. Está cláusula especifica que INSERTS e UPDATES executados pela view não têm permissão de criar linhas que a view não possa selecionar e, portanto, ela permite restrições de integridade e verificações de validação de dados a serem impostas aos dados que estiverem sendo inseridos ou atualizados. Se houver uma tentativa de executar operações DML em linhas que a view não selecionou, será exibido um erro, com o nome da restrição, se ele tiver sido especificado, por exemplo, a instrução DML a seguir realiza uma alteração no número do departamento, utilizando a view empvu20.

```
SQL> UPDATE empvu20
  2  SET    deptno = 10
  3  WHERE  empno = 7788;
update empvu20
        *
ERRO na linha 1: (ERROR at line1)
```

ORA-01402: violação na cláusula where WITH CHECK OPTION na view (view WITH CHECK OPTION where-clause violation)

Neste caso, nenhuma linha será atualizada porque se o número do departamento fosse alterado para 10, a view não seria mais capaz de enxergar o funcionário. Por isso, com a cláusula WITH CHECK OPTION, a view poderá ver apenas funcionários do departamento 20 e não permitirá que o número de departamento para esses funcionários seja alterado na view.

**Exemplo 5: Utilizando a cláusula WITH READ ONLY-** Alterar as definições da view empvu10 para que se torne somente para leitura.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2        (employee_number, employee_name, job_title)
  3  AS SELECT  empno, ename, job
  4  FROM                emp
  5  WHERE               deptno = 10
  6  WITH READ ONLY;
```

Quaisquer tentativas de inserir uma linha ou modificá-la usando a view resultará em erro no Oracle Server -01733: não é permitida coluna virtual aqui (virtual column not allowed here).

### 2.15.2. Removendo uma View

Pode-se remover uma view sem perder dados porque uma view está baseada em tabelas subjacentes no banco de dados.

**Sintaxe:**

```
DROP VIEW view;
```

**Exemplo:** Remover a view empvu10.

```
SQL> DROP VIEW empvu10;
```

A instrução DROP VIEW remove a definição da view do banco de dados. A eliminação de views não tem efeito nas tabelas nas quais ela é baseada. As views ou outras aplicações baseadas em views deletadas tornam-se inválidas. Apenas o criador ou usuário com o privilégio DROP ANY VIEW poderá remover uma view.

### 2.15.3. Consultando views no dicionário de dados

As definições da view ficam armazenadas no dicionário de dados como uma consulta. A tabela do dicionário de dados USER_VIEWS contém o nome e a definição da view. O texto da instrução SELECT que constitui a view é armazenado em uma coluna LONG.

**Exemplo:** Exibir o nome e o texto de definição das views do usuário corrente:

```
SQL> SELECT VIEW_NAME, TEXT FROM USER_VIEWS;
```

**Resultado:**

```
VIEW_NAME          TEXT
SALES             SELECT REPID, ORD.CUSTID, CUSTOMER.NAME CUSTNAME,
                  PRODUCT.PRODID, DESCRIP PRODNA
```

Quando a base de dados é acessada usando uma view, o Oracle Server executa as seguintes operações:

- Recupera a definição da view da tabela do dicionário de dados USER_VIEWS.
- Verifica os privilégios de acesso para a tabela-base da view.
- Converte a consulta da view em uma operação equivalente nas tabelas ou tabela-base subjacentes. Em outras palavras, os dados são recuperados a partir da(s) tabela(s)-base, ou uma atualização é feita nela(s).

## 2.16. Sequencia

Uma seqüência é um objeto do banco de dados criado pelo usuário que pode ser compartilhado por vários usuários para gerar números inteiros exclusivos. Pode-se usar as seqüências para gerar valores de chave primária automaticamente.
A seqüência é gerada e incrementada (ou diminuída) por uma rotina Oracle8 interna. Esse objeto pode economizar tempo, pois é capaz de reduzir a quantidade de código de aplicação necessária para criar uma rotina de geração de seqüências, além disso acelera a eficácia do acesso a valores de seqüência quando estão em cachê na memória.

### 2.16.1. Criando uma sequencia

**Sintaxe:**

```
CREATE SEQUENCE nome_seqüência
   [INCREMENT BY n]
   [START WITH n]
   [{MAXVALUE n | NOMAXVALUE}]
   [{MINVALUE n | NOMINVALUE}]
   [{CYCLE | NOCYCLE}]
   [{CACHE n | NOCACHE}];
```

onde:

| | |
|---|---|
| nome_*seqüência* | é o identificador da seqüência |
| INCREMENT BY *n* | especifica o intervalo entre números de seqüência onde *n* é um número inteiro, o default é 1. |
| START WITH *n* | especifica o primeiro número de seqüência a ser gerado, default é 1 |
| MAXVALUE *n* | especifica o valor máximo que a seqüência pode gerar |
| NOMAXVALUE | especifica um valor máximo de 10^27 para uma seqüência crescente e –1 para uma seqüência decrescente (NOMAXVALUE é a opção default) |
| MINVALUE *n* | especifica o valor de seqüência mínimo |
| NOMINVALUE | especifica um valor mínimo de 1 para uma seqüência crescente e – (10^26) para uma seqüência decrescente. (NOMINVALUE é a opção default) |
| CYCLE \| NOCYCLE | especifica que a seqüência continue a gerar valores após alcançar seu valor máximo ou mínimo ou não gere valores adicionais (NOCYCLE é a opção default.) |
| CACHE *n* \| NOCACHE | especifica quantos valores o Oracle Server alocará previamente e manterá na memória |

(Por default, o Oracle Server colocará em cache 20 valores.)

**Exemplo:** Criar uma seqüência identificada por dept_deptno. A seqüência deverá iniciar em 91, ser incrementada de 1 em 1, ter como valor máximo 100 e as demais configuarções default.

```
SQL> CREATE SEQUENCE dept_deptno
  2      INCREMENT BY 1
  3      START WITH 91
  4      MAXVALUE 100
  5      NOCACHE
  6      NOCYCLE;
```

O exemplo acima cria uma seqüência chamada DEPT_DEPTNO para ser usada na coluna DEPTNO da tabela DEPT. A seqüência começa em 91, não permite cachê e não permite o ciclo da seqüência.

**ATENÇÃO:** Se o valor do parâmetro INCREMENT By for negativo, a seqüência será descendente.

### 2.16.2. Consultando Seqüências no dicionário de dados.

As definições da seqüência ficam armazenadas no dicionário de dados e podem ser consultadas na view USER_SEQUENCES.

```
SQL>   select *
from user_sequences
where sequence_name = 'DEPT_DEPTNO';
```

**Resultado:**

```
SEQUENCE_NAME  MIN_VALUE   MAX_VALUE  INCREMENT_BY  C  O  CACHE_SIZE
LAST_NUMBER

DEPT_DEPTNO         1            3                1       Y  N       0
1
```

A coluna LAST_NUMBER exibe o próximo número de seqüência disponível.

### 2.16.3. Utilizando uma Seqüência

Para utilizar a seqüência é pode-se fazer referência aos seus valores usando as pseudocolunas NEXTVAL e CURRVAL.

- NEXTVAL retorna o próximo valor de seqüência disponível.
- CURRVAL obtém o valor de seqüência atual.

NEXTVAL e CURRVAL podem ser usadas nos seguintes casos:

- Na lista SELECT de uma instrução SELECT que não seja parte de uma subconsulta
- Na lista SELECT de uma subconsulta em uma instrução INSERT
- Na cláusula VALUES de uma instrução INSERT
- Na cláusula SET de uma instrução UPDATE

NEXTVAL e CURRVAL não podem ser usadas nos seguintes casos:

- Na lista SELECT de uma view
- Em uma instrução SELECT com a palavra-chave DISTINCT
- Em uma instrução SELECT com as cláusulas GROUP BY, HAVING ou ORDER BY
- Em uma subconsulta de uma instrução SELECT, DELETE ou UPDATE

**Exemplo 1: Utilizando NEXTVAL -** Inserir um registro na tabela dept utilizando a sequence dept_deptno para fornecer o código do departamento, o nome do departamento é MARKETING e a sua localização SAN DIEGO.

```
SQL> INSERT INTOdept(deptno, dname, loc)
  2  VALUES           (dept_deptno.NEXTVAL,
  3  'MARKETING', 'SAN DIEGO');
```

**Exemplo 2: Utilizando CURRVAL –** Consultar o próximo valor da seqüência dept_deptno:

```
SQL> SELECT     dept_deptno.CURRVAL
  2  FROM  dual;
```

### 2.16.3.1 Alterando uma Seqüência

É possível alterar as especificações de uma seqüência, para isso é necessário informar o cláusula que sofrerá alterações, as demais cláusulas permanecerão com os valores anteriores.

**Sintaxe:**

```
Alter SEQUENCE nome_seqüência
   [INCREMENT BY n]
   [{MAXVALUE n | NOMAXVALUE}]
        [{MINVALUE n | NOMINVALUE}]
   [{CYCLE | NOCYCLE}]
   [{CACHE n | NOCACHE}];
```

**Exemplo 3:** Aterar a sequence dept_deptno, o valor máximo deverá ser modificado para 99999 e as demais configuração deverão permanecer as mesmas.

```
SQL> ALTER SEQUENCE dept_deptno
 2     MAXVALUE 999999;
```

**ATENÇÃO:**

- Somente o proprietário ou usuários que possuam o privilégio ALTER podem alterar a seqüência.
- Somente os números de seqüência futuras são afetados.
- A seqüência deve ser eliminada e recriada para reiniciar a seqüência em um número diferente.
- Alguma validação é executada, Por exemplo, não é possível impor um novo MAXVALUE menor do que o número de seqüência atual.

### 2.16.4. Removendo uma Seqüência

Após remover a seqüência do dicionário de dados, não será possível fazer referência à ela.

**Sintaxe:**

```
DROP SEQUENCE nome_seqüência;
```

**Exemplo:** Remover a sequencia dept_deptno.

```
SQL> DROP SEQUENCE dept_deptno;
```

**ATENÇÃO:** Somente o proprietário da seqüência ou usuários com privilégio DROP ANY SEQUENCE podem remover uma seqüência.

## 2.17. Sinônimo

Um sinônimo é um nome alternativo para um objeto do banco de dados. Este recurso pode ser empregado para facilitar o acesso aos objetos ou para esconder a verdadeira identidade de um objeto. Por exemplo, a tabela EMP faz parte do esquema do usuário Scott, supondo que um outro usuário, que tenha privilégios, utilize está tabela frequentemente para realizar consulta, para isso precisará fazer referência ao esquema e a instrução seria apresentada da seguinte maneira: `Select * from Scott.emp;` com uso de um sinônimo para fazer referencia ao objeto, este poderia ser tratado por um nome alternativo, por exemplo empregado, desta forma qualquer usuário que necessite fazer uso da tabela emp do usuário Scott, poderia fazê-lo utilizando o sinônimo empregado: `Select * from empregado;` Internamente o Oracle identifica e localiza o objeto pela referencia original e completa.

**Sintaxe:**

```
CREATE [PUBLIC] SYNONYM sinônimo
FOR    objeto;
```

onde:

| | |
|---|---|
| PUBLIC | cria um sinônimo acessível a todos os usuários |
| *sinônimo* | é o nome do sinônimo a ser criado |
| *objeto* | identifica o objeto para o qual o sinônimo será criado |

**ATENÇÃO:**

- O objeto que receberá um sinônimo não pode estar contido em um pacote.
- Um nome de sinônimo deve ser distinto de todos os outros objetos de propriedade do mesmo usuário.

**Exemplo**: Criar um sinônimo de uso público para a tabela emp do usuário Scott.

```
SQL> CREATE PUBLIC SYNONYM empregado
  2  FOR      Scott.emp;
```

#### 2.17.1. Consultando Sinônimos no dicionário de dados

As definições dos sinônimos ficam armazenadas no dicionário de dados e podem ser consultadas na view USER_SYNONYM.

**Exemplo:** Consultar todas as definições dos sinônimos do usuário corrente.

```
SQL> select * from user_synonyms;
```

**Resultado:**

```
SYNONYM_NAME     TABLE_OWNER        TABLE_NAME     DB_LINK
------------------------------ ------------------------------ ----
CATALOG          SYS                 CATALOG
COL              SYS                 COL
```

#### 2.17.2. Removendo um Sinônimo

A instrução para se remover um sinônimo é:

```
DROP SYNONYM nome_sinônimo;
```

**Exemplo**: Eliminar o sinônimo empregado.

```
SQL> DROP SYNONYM empregado;
```

**ATENÇÃO:** Somente usuários com privilégios de DBA podem eliminar um sinônimo público

## 2.18. Índice (Index)

Um índice do *Oracle Server* é um objeto de esquema que pode acelerar a recuperação de linhas usando um ponteiro. Se não houver um índice na coluna, ocorrerá uma análise em toda a tabela.
O índice é muito parecido com uma chave:

- Permite classificação por outros campos
- Acelera a busca de registros específicos
- A chave primária e a chave secundária são índices

A diferença entre chave e índice é que as chaves são estruturas lógicas, usadas para identificar registros em uma tabela, e índices são estruturas físicas, usadas para otimizar o processamento de dados pois são objetos no banco de dados que fornecem um mapeamento de todos os valores em uma coluna de uma tabela.
Os índices podem ser usados para garantir a unicidade dos elementos inseridos numa coluna (ou campo) e também para alavancar a performance na busca por registros.
O aumento de performance é obtido quando o critério de pesquisa por dados na tabela inclui referência de coluna(s) indexadas.
Um índice fornece acesso direto e rápido às linhas em uma tabela. Seu objetivo é reduzir a necessidade de E/S do disco usando um caminho indexado para localizar dados rapidamente. O índice é usado e mantido automaticamente pelo Oracle Server. Após a criação de um índice, não é necessária nenhuma atividade direta do usuário.
Os índices são lógica e fisicamente independentes da tabela que indexam. Isso significa que eles podem ser criados e eliminados a qualquer momento e não têm nenhum efeito sobre as tabelas-base ou outros índices.

**ATENÇÃO:** Quando uma tabela é eliminada, os índices associados a ela também são eliminados.

É possível criar dois tipos de índices:

- índice exclusivo. O Oracle Server cria esse índice automaticamente quando você define que uma coluna de uma tabela tenha uma restrição PRIMARY KEY ou UNIQUE KEY. O nome do índice é o nome dado à restrição.
- índice não-exclusivo. Por exemplo, você pode criar um índice da coluna FOREIGN KEY para uma junção em uma consulta a fim de aumentar a velocidade de recuperação.

### 2.18.1. Criando um Índice

**Sintaxe:**

```
CREATE INDEX nome_índice ON tabela (coluna[, coluna]...);
```

onde:

| | |
|---|---|
| nome_*índice* | é o nome do índice |
| *tabela* | é o nome da tabela |
| *coluna* | é o nome da coluna na tabela a ser indexada |

**Exemplo:** Criar um índice para coluna ename da tabela emp.

```
SQL> CREATE INDEX      emp_ename_idx
  2  ON                emp(ename);
```

É possível criar vários índices para uma tabela, mas isso não significa que as consultas serão aceleradas. Cada operação DML que seja submetida a commit em uma tabela com índices significa que os índices devem ser atualizados. Quanto mais índices associados a uma tabela você tiver, maior será o esforço feito pelo Oracle Server para atualizar todos os índices após uma DML. Por isso recomenda-se a criação de índices quando:

- a coluna for usada freqüentemente na cláusula WHERE ou em uma condição de junção.
- a coluna contiver uma ampla faixa de valores.
- a coluna contiver um grande número de valores nulos.
- duas ou mais colunas forem usadas juntas com freqüência em uma cláusula WHERE ou em uma condição de junção.
- a tabela for grande e se esperar que a maioria das consultas recupere menos que 2 a 4% das linhas.

**ATENÇÃO:**

- Lembre-se de que, para aplicar exclusividade, deve-se definir uma restrição exclusiva na definição da tabela. Em seguida, um índice exclusivo será criado automaticamente.
- Quando a tabela for atualizada com freqüência. Se você tiver um ou mais índices em uma tabela, as instruções DML que acessarem a tabela serão mais lentas.

### 2.18.2. Consultando Índices no dicionário de dados.

As definições do índice ficam armazenadas no dicionário de dados, é possível consultá-las utilizando a view USER_INDEXES. Também é possível checar as colunas envolvidas em um índice consultando a view USER_IND_COLUMNS.

**ATENÇÃO:** A view user_indexes contém muitas colunas, por isso é recomendado que verifique estrutura da view e elabore uma consulta somente com as colunas relevantes à pesquisa, para isso utilize a instrução DESCRIBE user_indexes

**Exemplo:** Exibir as informações sobre o tipo de índice e o nome da tabela para o índice emp_ename_idx.

```
  1  SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME
  2  FROM USER_INDEXES
  3* WHERE INDEX_NAME = 'EMP_ENAME_IDX'
```

**Resultado:**

```
INDEX_NAME                INDEX_TYPE                   TABLE_NAME
---------------------  ---------------------------  ----------------
EMP_ENAME_IDX             NORMAL                       EMP
```

### 2.18.3. Removendo um Índice

Somente o proprietário do índice ou usuários com privilégio  DROP ANY INDEX. podem remover o índice.

**Sintaxe:**

```
      DROP INDEX nome_indice
```

**Exemplo:** Eliminar o índice emp_ename_idx.

```
SQL> DROP INDEX nome_indice
```

**ATENÇÃO:** Um índice  não pode ser modificado, para altera-lo deve-se eliminá-lo e, em seguida, recriá-lo.

## 2.19. Usuário

Existem dois tipos de usuário que podem acessar o banco: os usuários que se autenticam pelo arquivo de senhas e os usuários que se autenticam pelo dicionário de dados.
Os usuários que se autenticam pelo arquivo de senhas são os usuários DBA com privilégios para  realizar startup e shutdown no banco.
Os usuários que se autenticam pelo dicionário de dados podem possuir privilégios de DBA mas não podem realizar startup e shutdown no banco. Estes usuários são criados e seu perfil fica armazenado no dicionário de dados, sendo assim ele poderá realizar apenas as atividades descritas no seu perfil.
A criação de um usuário é feita com uma instrução DDL e as especificações são armazenadas no dicionário de dados, por isso também é tratado como um objeto do banco.

**Sintaxe** Simplificada:

```
CREATE USER nome_usuário
IDENTIFIED BY  senha PASSWORD EXPIRE DEFAULT;
```

onde:

| | |
|---|---|
| `nome_usuário` | nome do usuário |
| `identified by` | atribui uma senha para o usuário |
| `senha` | senha que será atribuida ao usuário, não deve iniciar com números |
| `PASSWORD EXPIRE DEFAULT` | quando o usuário se conecta ao banco ele deve redefinir a sua senha de autenticação. Este recurso é válido somente para os usuários que se autenticam pelo DD |

**Exemplo:** Criar o usuário EXEMPLO, com senha PBD cuja senha deverá ser modificada quando a primeira conexão for realizada.

```
SQL> CREATE USER EXEMPLO
  2  IDENTIFIED BY "PBD"
  3  PASSWORD EXPIRE;
```

**ATENÇÃO:** Para que o usuário efetue a conexão é necessário conceder-lhe privilégios de conexão (este tópico será visto posteriormente).

#### 2.19.1. Alterando usuários

As definições atribuídas a um usuário podem ser alteradas, isso pode ser feito pelo DBA ou por usuários com privilégio ALTER USER. Para isso deve ser informado o nome do usuário, o parâmetro que deverá ser alterado e o valor do novo parâmetro.

**Sintaxe:**

`ALTER USER` *usuário* `IDENTIFIED BY` *senha*;

onde:

| | |
|---|---|
| *usuário* | é o nome do usuário |
| *senha* | especifica a nova senha |

**Exemplo:** Alterar a senha do usuário scott para lion.

```
SQL> ALTER USER scott
2    IDENTIFIED BY lion;
```

#### 2.19.2. Removendo usuários

**Sintaxe:**

```
drop user nome_usuário;
```

**Exemplo:** Eliminar o usuário exemplo.

```
SQL> drop user exemplo;
```

**ATENÇÃO:** Um esquema é uma coleção de objetos como, por exemplo, tabelas, views e seqüências. O esquema pertence a um usuário de banco de dados e tem o mesmo nome do usuário.

#### 2.19.3. Personagens [16]

Pode-se criar um objeto do banco denominado personagem, atribuir privilégios a este personagem e depois atribuir estes privilégios a um usuário. Isso faz com que a concessão e revogação de privilégios se torne mais fácil de desempenhar e manter.

**Sintaxe:**

```
CREATE   ROLE  nome_personagem;
```

onde:

personagem é o nome do personagem ou papel a ser criado

**Exemplo:**

```
SQL> CREATE ROLE manager;
```

Depois de criado um personagem é necessário atribuir os privilégios a ele.

### 2.20. Privilégios

Os privilégios são os direitos concedidos aos usuários para executar instruções SQL particulares. Os privilégios podem ser de sistema ou objetos.
O administrador de banco de dados é um usuário de alto nível que pode conceder aos usuários acesso ao banco de dados e aos objetos. Os usuários requerem *privilégios de sistema* para obter acesso aos *privilégios de objeto* e de banco de dados para manipular o conteúdo dos objetos no banco de dados. Também pode ser fornecido aos usuários o privilégio de conceder privilégios adicionais a outros usuários ou a *funções*, que são grupos nomeados de privilégios relacionados.

#### 2.20.1. Privilégios de sistema

Os privilégios de sistema permitem que os usuários executem determinadas ações no banco de dados.

Exemplos de privilégios de sistema:

---

[16] A Oracle como funções, para evitar confusões será utilizado o termo personagem também utilizado por Abbey & Corey em Oracle – Guia do usuário

| | |
|---|---|
| INDEX | CREATE ANY INDEX |
| | ALTER ANY INDEX |
| | DROP ANY INDEX |
| TABLE | CREATE TABLE |
| | CREATE ANY TABLE |
| | ALTER ANY TABLE |
| | DROP ANY TABLE |
| | SELECT ANY TABLE |
| | UPDATE ANY TABLE |
| | DELETE ANY TABLE |
| SESSION | CREATE SESSION |
| | ALTER SESSION |

**Sintaxe:**

GRANT {privilégio | role} ...
TO {usuário | role | public} ...
[WITH ADMIN OPTION];

onde:

| | |
|---|---|
| *privilégio* | é o privilégio de sistema a ser concedido |
| *usuário* | *é o n*ome do usuário |
| *role* | é um personagem (função ou papel) |
| public | concede os privilégios de sistema a todos os usuários |
| *WITH ADMIN OPTION* | *permite que o usuário que está recebendo um privilégio ou as atribuições possa concede-las a outros usuários ou atribuições.* |

***Exemplo 1:*** *Conceder privilégios de conexão para o usuário exemplo:*

```
SQL> grant connect to exemplo;
```

***Exemplo 2:*** *Conceder privilégios para criar tabelas, views e sequences para o usuário scott, que poderá conceder estes privilégios a outros usuários.*

```
SQL> SQL> GRANT  create table, create sequence, create view
  2  TO     scott WITH ADMIN OPTION;
```

### 2.20.2. Privilégios de objeto

Os privilégios de objeto permitem que os usuários acessem e manipulem objetos específicos, por exemplo inserir linhas em uma tabela de outro usuário.
Alguns Privilégios de objeto:

| Privilégio | Tabela | Visão | Seqüência | Procedimento |
|---|---|---|---|---|
| ALTER | x | | x | |
| DELETE | x | x | | |
| EXECUTE | | | | x |

| INDEX | x | | | |
|---|---|---|---|---|
| INSERT | x | x | | |
| REFERENCES | x | | | |
| SELECT | x | x | x | |
| UPDATE | | | | |

**Sintaxe:**

```
GRANT [privilégio  (lista de colunas) | ALL PRIVILEGES} ...
ON esquema.objeto
TO {usuário | role | public} ...
[WITH GRANT OPTION];
```

onde:

| | |
|---|---|
| *privilégio* | é o privilégio de objeto a ser concedido |
| lista de colunas | especifica as colunas de uma view ou tabela |
| on esquema.objeto | especifica o nome do objeto |
| *to usuário* | é o nome do usuário |
| *role* | é um personagem ou função |
| *public* | concede os privilégios de sistema a todos os usuários |
| *WITH GRANT OPTION* | permite que o usuário conceda o privilégio recebido a outro usuário |

**Exemplo 1:** Conceder ao usuário exemplo o privilégio de inserir dados nas colunas empno, ename e deptno da tabela EMP do usuário scott.

```
SQL>  GRANT INSERT(empno, ename, deptno) on SCOTT.EMP
to exemplo;
```

**Exemplo 2:** Conceder ao usuário exemplo o privilégio de excluir e alterar dados na tabela EMP do usuário scott, podendo inclusive repassar estes privilégios a outros usuários.

```
SQL>  SQL>  GRANT DELETE, UPDATE
  2   ON SCOTT.EMP
  3   TO EXEMPLO
  4   WITH GRANT OPTION;
```

### 2.20.3. Concedendo privilégios por meio de personagens

**Exemplo:** Conceder o privilégio de criar tabelas e visões para o personagem manager e depois atribuir este personagem aos usuários BLAKE e CLARK.

```
SQL> GRANT create table, create view
2 to manager;

SQL> GRANT manager to BLAKE, CLARK;
```

**ATENÇÃO:**

- Para conceder privilégios sobre um objeto, ele deve estar no esquema do próprio usuário ou então o usuário deve ter recebido os privilégios de objeto WITH GRANT OPTION.
- Um proprietário de objeto pode conceder qualquer privilégio de objeto sobre o objeto para qualquer outro usuário ou personagem do banco de dados.
- O proprietário de um objeto adquire automaticamente todos os privilégios de objeto sobre seus objetos.

#### 2.20.4. Revogando privilégios

**Sintaxe:**

```
REVOKE privilégios FROM usuário [cascade contraints];
```

onde,

| | |
|---|---|
| `REVOKE` | comando para revogar privilégios |
| `privilégios` | lista de privilégios a serem revogados, devem ser separados por vírgula |
| `FROM usuário|role` | indica o usuário ou role do qual o privilégio será removido |
| `cascade constraints` | elimina as restrições de integridade de referência definidas pela revogação usando os privilégios REFERENCES ou ALL. |

**Exemplo 1 : Revogando privilégios de sistema -** Revogar os privilégios de create table, create sequence e create view do usuário exemplo.

```
SQL> revoke create table, create sequence, create view
from exemplo;
```

**Exemplo 2 : Revogando privilégios de objeto -** Revogar os privilégios de exclusão de linhas na tabela SCOTT.EMP do usuário exemplo.

```
SQL> revoke delete
  2  on SCOTT. EMP
from exemplo;
```

**ATENÇÃO:** Privilégios de sistema não são revogados em cascata.
Privilégios de objeto são revogados em cascata.

#### 2.20.5. Consultado os privilégios

Pode-se consultar os privilégios que um usuário ou personagem possui, para isso pode-se utilizar as tabelas do dicionário de dados, a seguir são apresentadas algumas possibilidades.

| Tabela de Dicionário de Dados | Descrição |
|---|---|

| | |
|---|---|
| ROLE_SYS_PRIVS | Privilégios de sistema concedidos a personagens |
| ROLE_TAB_PRIVS | Privilégios de tabela concedidos a personagens |
| USER_COL_PRIVS_MADE | Os privilégios de objeto concedidos às colunas dos objetos do usuário |
| USER_COL_PRIVS_RECD | Os privilégios de objeto concedidos ao usuário em colunas específicas |
| USER_ROLE_PRIVS | Personagens acessíveis ao usuário |
| USER_TAB_PRIVS_MADE | Os privilégios de objeto concedidos aos objetos do usuário |
| USER_TAB_PRIVS_RECD | Os privilégios de objeto concedidos ao usuário |

**Exemplo:** Exibir a lista de privilégios de objeto concedidos às colunas dos objetos do usuário.

```
SQL> select * from USER_COL_PRIVS_MADE;
```

**Resultado**

```
GRANTEE                       TABLE_NAME                    COLUMN_NAME           GRANTOR
----------------------------- ----------------------------- --------------------- --------------
EXEMPLO                       EMP                           EMPNO                 SCOTT
EXEMPLO                       EMP                           ENAME                 SCOTT
EXEMPLO                       EMP                           DEPTNO                SCOTT
```

**Para saber mais consulte:**

| **Livro** | **Comentários** |
|---|---|
| | Este livro aborda. |

# 3. PL/SQL

A linguagem PL/SQL (Procedural Language/SQL) é uma extensão de linguagem procedural da Oracle Corporation para SQL, incorpora muitos recursos de programação, tais como:

- declaração de variávies e tipos pré-definidos
- Estruturas de seleção e repetição
- Procedimentos e funções
- Tipos de objeto e métodos (nas versões posteriores a 8)

Além de aceitar a manipulação de dados, ela também permite que as instruções de consulta da linguagem SQL sejam incluídas em unidades procedurais de código e estruturadas em blocos, tornando a linguagem SQL uma linguagem avançada de processamento de transações. Com a linguagem PL/SQL, é possível criar blocos de anônimos ou blocos nomeados[17].
.A linguagem PL/SQL é estruturada em blocos, os programas podem ser divididos em blocos lógicos. Um bloco PL/SQL é dividido em três seções:

**DECLARATIVA (opcional)** - Seção destinada à declaração das variáveis, cursores e exceções que serão utilizadas no bloco
**EXECUTÁVEL (obrigatória)** - também conhecida como corpo do programa, área onde são descritos os passos necessários para realização da tarefa. Podem ser utilizadas instruções SQL e/ou PL/SQL
**TATAMENTO DE EXCEÇÕES (opcional) -** destinada ao tratamento das exceções geradas no bloco; devem ser descritas as ações a serem desempenhadas quando ocorrerem erros.

## 3.1. Blocos Anônimos

Os blocos anônimos não ficam armazenados na base de dados e não podem ser chamados por outros blocos PL/SQL, eles devem ser compilados a cada utilização. Pode-se incorporar um bloco anônimo em uma aplicação ou pode-se executá-lo interativamente no SQL*Plus.
Exemplo da estrutura de um bloco em PL/SQL:

```
DECLARE
   v_variavel varchar2(5);
BEGIN
   Select  nome_coluna
   into v_variavel
```

[17] Os blocos nomeados (procedimentos, funções, triggers e packages) possuem algumas particularidades que serão estudadas no 4°. semestre, são compilados e armazenados na base de dados e por isso não precisam ser compilados a cada execução

```
   from nome_tabela;
EXCEPTION
   When exception_name then
    ... procedimentos a serem executados quando uma determinada exceção ocorrer.
END;
```

Sobre o código:

- As palavras chave DECLARE, BEGIN e EXCEPTION não são sucedidas de ponto e vírgula, as demais instruções são terminadas com um ponto e vírgula (;).
- A palavras-chave BEGIN e END são obrigatórias.
- Deve-se utilizar uma barra (/) para executar o bloco anônimo PL/SQL no buffer de SQL*Plus.
- Coloque um ponto (.) para fechar um buffer de SQL*Plus. Um bloco PL/SQL é tratado como uma instrução contínua no buffer e os ponto-e-vírgulas no bloco não fecham ou executam o buffer.
- A edição do bloco pode ser feita utilizando um editor de textos cujo arquivo deverá possuir extensão sql, por exemplo teste.sql. No SQL*Plus é possível definir o editor default, para isso deve-se selecionar, no menu, o item Editor – Definir Editor.

### 3.1.1. Operadores e delimitadores

| Operador | Descrição |
| --- | --- |
| + | operador de adição |
| * | operador de multiplicação |
| = | operador de igualdade |
| > | operador de comparação – maior do que |
| ) | delimitador de fim expressão |
| % | indicador de atributo |
| . | separador de componentes |
| ‘ | delimitador de cadeia de caracteres |
| : | indicador de variável de ligação |
| ~=<br>^= | operador de comparação – diferente |
| := | operador de atribuição |
| .. | operador de intervalo |
| -- | indicador de comentário de linha |
| /* | delimitador de início de comentário de linha |
| << | delimitador de início do texto do label(etiqueta) |

| Operador | Descrição |
| --- | --- |
| - | operador de subtração |
| / | operador de divisão |
| <><br>!= | operador de comparação – diferente |
| < | operador de comparação – menor do que |
| ( | delimitador de início de expressão |
| ; | terminador de instrução |
| , | separador de itens |
| “ | delimitador de cadeia de caracteres (mensagens) |
| ** | operador de exponenciação |
| \|\| | operador de concatenação |
| | |
| >= | operador de comparação – maior ou igual |
| <= | operador de comparação – menor ou igual |
| */ | delimitador de fim de comentário de linha |
| >> | delimitador de final do texto do label(etiqueta) |

### 3.1.2. Variáveis

Os dados podem ser armazenados temporariamente em uma ou mais variáveis e podem ser utilizadas para:

**Manipulação de valores armazenados -** As variáveis podem ser usadas para cálculo e manipulação de outros dados sem acessar o banco de dados, ou seja, após os valores em memória não há necessidade de outros acessos para complemento da informação já armazenada.
**Reutilização -** Quando declaradas, podem ser usadas repetidamente em uma aplicação simplesmente referenciando-as em outras instruções, incluindo outras instruções declarativas.
**Facilitar a manutenção –** Pode-se declarar variáveis baseadas na estrutura das colunas das tabelas ou em outras variávies (%TYPE e %ROWTYPE). Se uma definição subjacente for alterada, a declaração da variável é atualizada em tempo de execução. Isso permite a independência dos dados, reduz custos de manutenção e permite que os programas se adaptem de acordo com as alterações realizadas no banco de dados
**Passar valores** aos subprogramas PL/SQL através de parâmetros.
**Exibir os resultados** em um bloco PL/SQL através de variáveis de saída.

A declaração e a inicialização das variáveis é feita na seção declarativa de qualquer subprograma pacote ou bloco PL/SQL. As declarações alocam espaço de armazenamento para um valor, especificam seus tipos de dados e nomeiam a localização de armazenamento para que se possa referenciá-los. As declarações poderão também atribuir um valor inicial e impor a restrição NOT NULL.

**ATENÇÃO:** Ao Atribuir novos valores às variáveis na seção executável o valor existente da variável é substituído pelo novo e é necessário declarar uma variável antes de referenciá-la em outras instruções, incluindo outras instruções declarativas.

### 3.1.3. Tipos de Variáveis

Todas as variáveis PL/SQL têm um tipo de dados, o qual especifica um formato de armazenamento, restrições e uma faixa válida de valores. A linguagem PL/SQL suporta quatro categorias de tipos de dados[18]:

**Escalares -** armazenam um único valor. Os principais tipos de dados são aqueles que correspondem aos tipos de coluna nas tabelas do Oracle Server, por exemplo vachar2, number, char, entre outros; a linguagem PL/SQL também suporta variáveis booleanas.

[18] Neste curso a ênfase será nos tipos escalares, variáveis de ligação e o tipo composto registro.

**Compostos** – comporta o armazenamento de diferentes valores. Os tipos compostos em PL/SQL são registro, tabelas e matrizes.
**Referenciais -** armazenam valores, chamados de indicadores, que designam outros itens de programa. Um exemplo de tipos referenciais é o REF CURSOR.
**LOB (**large object) - armazenam blocos de dados não estruturados (como, por exemplo, texto, imagens gráficas, videoclipes e formatos de arquivo para armazenar sons) de até 4 gigabytes em tamanho. Os tipos de dados LOB fornecem acesso eficiente, aleatório e em intervalos aos dados, podendo ser atributos de um tipo de objeto. As variáveis LOB podem ser classificadas como:
O tipo de dados CLOB (character large object, objeto grande de caractere) é usado para armazenar blocos grandes de dados com caracteres de um único byte no banco de dados.
O tipo de dados BLOB (binary large object, objeto grande binário) é usado para armazenar objeto binários grandes no banco de dados em linha (dentro de uma linha de tabela) ou fora de linha (fora da linha de tabela).
O tipo de dados BFILE (binary file, arquivo binário) é usado para armazenar objetos grandes binários em arquivos do sistema operacional fora do banco de dados.
O tipo de dados NCLOB (objeto grande de caractere do idioma nacional) é usado para armazenar blocos grandes de dados NCHAR de byte único ou de bytes múltiplos de largura fixa no banco de dados, dentro e fora de linha.

**Declaração de Variáveis**

**Sintaxe:**

```
identificador [CONSTANT] tipo de dados [NOT NULL]
        [:= valor para inicialização | expr default]
```

onde,

| | |
|---|---|
| dentificador | é o nome da variável; Os identificadores não devem ter mais de 30 caracteres. O primeiro caracter deve ser uma letra; os demais podem ser letras, números ou símbolos especiais; Não devem ser palavras reservadas nem possuir espaços entre os caracteres. |
| CONSTANT | restringe as variáveis para que o seu valor não possa ser alterado; as constantes devem ser inicializadas |
| tipo de dados | são tipos de dados escalares, compostos, referenciais ou LOB |
| NOT NULL | indica preenchimento obrigatório (variáveis NOT NULL devem ser inicializadas.) |
| expr | é uma expressão PL/SQL que pode ser uma literal, uma outra variável ou uma expressão que envolve operadores e funções |

**ATENÇÃO:**

- Para inicializar a variável utiliza-se operador de atribuição (:=) ou a palavra reservada DEFAULT. Se não for atribuído um valor inicial, a nova variável conterá NULL por default.
- Não é aconselhável identificar uma variável com nome igual ao nome das colunas de tabela usadas no bloco. Se as variáveis PL/SQL ocorrerem nas instruções SQL e tiverem o mesmo nome que uma coluna, o Oracle Server supõe que seja a coluna que esteja sendo referenciada.

#### 3.1.3.1 Tipos de Dados Escalares

Um tipo de dados escalares armazena um valor único e não possui componentes internos. Os tipos de dados escalares podem ser classificados em quatro categorias: número, caractere, data e booleano. Os tipos de dados de caractere e número possuem subtipos que associam um tipo básico a uma restrição. Por exemplo, INTEGER e POSITIVE são subtipos do tipo básico NUMBER.
Segue abaixo exemplo de tipos de Dados Escalares Básicos:

| Tipo de dado | Descrição |
|---|---|
| VARCHAR2(tamanho) | Armazena caracteres com tamanho variável com até 32.767 bytes; não há tamanho default para estas variáveisl |
| NUMBER(tamanho, precisão) | Armazena números reais ou inteiros |
| DATE | Armazena datas e horas entre os períodos de 4712 A.C. e 9999 D.C. |
| CHAR(tamanho) | Armazena caracteres de tamanho fixo até 32.767 bytes;tamanho default 1 byte. |
| LONG | Armazena caracteres com tamanho variável de até 32.760 bytes, a largura máxima é de 2.147.483.647 bytes |
| LONG RAW | Armazena dados binários e strings de byte de até 32.760 bytes; estes dados não são interpretados pelo PL/SQL |
| BOOLEAN | Armazena um de três possíveis valores: TRUE, FALSE ou NULL |
| BINARY_INTEGER | Armazena números inteiros entre –2.147.483.647 e 2.147.483.647 |
| PLS_INTEGER | Armazena números inteiros entre –2.147.483.647 e 2.147.483.647; estes valores requerem menos armazenamento e são mais rápidos que os valores NUMBER e BINARY_INTEGER |

**Exemplos:**

```
v_nascimento date;

v_data date := sysdate + 7; --declaração de variável e inicialização à partir de uma operação aritmética
```

```
v_codigo number(2) not null := 10; -- declaração da variável com a restrição de preenchimento obrigatório, neste caso é preciso atribuir o valor inicial

v_UF varchar2(2) := 'SP'; -- declaração e inicialização da variável com o valor SP; observe que os literais devem ser informados entre aspas simples (' ')

v_Loc varchar2(2) default 'RJ'; -- declaração da variável cujo valor default é RJ.

v_teste_logico boolean := (v_valor1 < v_valor2); -- declaração da variável v_teste_logico que será inicializada com o resultado da expressão (v_valor1 < v_valor2)

c_const constant number := 54; -- declaração da constante que está sendo inicializada.
```

O atributo `%TYPE` pode ser utilizado para declarar variáveis de com a mesma estrutura de uma tabela ou de uma variável já existente.

**Sintaxe:**

```
identificador [CONSTANT] {tabela.coluna%type | variavel%type}
[NOT NULL] [:= valor para inicialização | expr default]
```

**Exemplos:**

```
v_nome emp.ename%type; -- declaração da variável com a mesma estrutura da coluna ename da tabela emp.

v_balance number(7,2);

v_min_balance v_balance%type; -- declaração da variável com a mesma estrutura da variável declarada anteriormente.
```

**ATENÇÂO:** É uma boa prática de programação a adoção de uma convenção de nomeação para variáveis, por exemplo: o prefixo v_ representa uma variável; c_ representa uma constante.

#### 3.1.4. Variáveis de Substituição

Além das variáveis declaradas no Bloco PL/SQL existem as variáveis de substituição ou de ligação. O código PL/SQL não tem capacidade de entrada/saída própria, para isso pode-se utilizar o ambiente no qual o código

PL/SQL estiver sendo executado para passar valores para o bloco, estes valores são passados por meio das variáveis de substituição.

**ATENÇÃO:** Ao utilizar as variáveis de substituição os valores são substituídos no bloco PL/SQL antes que esse bloco seja executado. Por isso, não é possível substituir os diferentes valores para as variáveis de substituição usando um loop.

**Exemplo:**

ACCEPT p_nome PROMPT 'Digite o nome do funcionário...'
ACCEPT p_sal_mes PROMPT 'Digite o valor do salario...'

O comando accept indica que o valor para a variável de substituição p_nome e p_sal_mes serão fornecidos pelo teclado. O comando PROMPT indica a mensagem que será exibida para orientar o usuário quanto ao valor a ser digitado. Para atribuir os valores das variáveis de substituição para as variáveis PL/SQL é necessário a utilização do &.

**Exemplo:**
Declare
    v_nome varchar2(30) := '&p_nome';
    v_sal number(9,2) := &p_sal_mes;

#### 3.1.5. Variáveis de ligação

Para declarar uma variável de ligação no ambiente SQL*Plus, utiliza-se o comando VARIABLE.

**Exemplo:**

```
SQL> VARIABLE g_sal_anual NUMBER
```

Estas variáveis podem ser declaradas no prompt do SQL ou fora do bloco PL/SQL (antes d o declare ou depois da /).

Para utilizar estas variáveis no bloco PL/ SQL deve-se precede-las de : (dois pontos), para que sejam distinguidas das variáveis PL/SQL:

**Exemplo:**

```
:g_sal_anual := v_sal * 12;
```

A exibição dos valores armazenados nestas variáveis pode ser feita por meio do comando SQL PRINT:

**Exemplo:**

```
PRINT g_sal_anual
```

**Exemplo 1 (programa completo):** Calcular o salário anual de dado funcionário com base no salário mensal fornecido pelo usuário. Exibir o resultado.

| | |
|---|---|
| VARIABLE g_sal_anual NUMBER<br>ACCEPT p_nome PROMPT 'Digite o nome do funcionário...'<br>ACCEPT p_sal_mes PROMPT 'Digite o valor do salario...' | **Instruções SQL** |
| Declare<br>      v_nome varchar2(30) := '&p_nome';<br>      v_sal number(9,2) := &p_sal_mes;<br>**Begin**<br>  :g_sal_anual := v_sal * 12;<br>**END;**<br>**/** | Bloco PL/SQL |
| **PRINT** g_sal_anual | **Instruções SQL** |

Para testar o exemplo 1:

escreva o código utilizando um editor d textos, salve o arquivo com extensão sql, por exemplo: exemplo1.sql.
no prompt do sql digite: @localização do arquivo\nome do arquivo, por exemplo: SQL>@ C:\programas\exemplo1.sql

No exemplo 1 foi utilizado o comando SQL PRINT para exibição do resultado do processamento, uma outra opção para isso é a utilização do pacote DBMS_output e procedimento put_line.

**Sintaxe:**

```
DBMS_output.put_line(mensagem);
```

onde,
mensagem pode ser:

- uma cadeia de caracteres, por exemplo: DBMS_output.put_line('O nome do funcionário é'); e neste caso deverá ser especificada entre aspas simples.
- uma variável ou constante, por exemplo: DBMS_output.put_line(v_sal_anual);
- ou ambas, por exemplo: DBMS_output.put_line(nome do funcionário é ' || v_sal_anual); neste caso está sendo utilizado o operador de concatenação || para unir as duas mensagens.

**ATENÇÃO:** Para a utilização deste pacote é necessário ativá-lo no SQL*PLUS por meio da instrução SET SERVEROUTPUT ON.

**Exemplo 2 (programa completo):** Calcular o salário anual de dado funcionário com base no salário mensal fornecido pelo usuário. Exibir o resultado.

```
SET SERVEROUTPUT ON
ACCEPT p_nome PROMPT 'Digite o nome do funcionário...'          Instruções
ACCEPT p_sal_mes PROMPT 'Digite o valor do salario...'          SQL

Declare
      v_nome varchar2(30) := '&p_nome';
      v_sal number(9,2)   := &p_sal_mes;
      v_sal_anual number(9,2);                                  Bloco
Begin                                                           PL/SQL
  v_sal_anual := v_sal * 12;
  DBMS_output.put_line('O salario anual de ' ||
v_nome || ' é ' || v_sal_anual);
END;
/
```

Note que no exemplo 2 não está sendo utilizada a variável de ligação g_sal_anual, pois a solicitação da exibição do resultado do processamento está sendo feita na área executável do bloco PL/SQL.

### 3.1.6. Instruções SELECT em PL/SQL

Recuperar dados do banco de dados com SELECT.

**Sintaxe:**

```
SELECT  colunas
INTO   {variáveis...| registro}
FROM   tabela
WHERE  condição;
```

onde,

| | |
|---|---|
| *colunas* | lista de colunas que retornarão dados à consulta; podem incluir funções de linhas, de grupo ou expressões SQL |
| *into* | cláusula obrigatória, usada para especificar os nomes das variáveis que armazenarão os valores que o SQL retornará a partir da cláusula SELECT. Deve-se oferecer uma variável para cada coluna, seguindo a mesma ordem. |

| | |
|---|---|
| *variáveis* | nome das variávies que irão armazenar o valor recuperado |
| *registro* | é o registro PL/SQL para armazenar os valores recuperados |
| *tabela* | *e*specifica o nome da tabela do banco de dados |
| *condição* | é composta de nomes de coluna, expressões, constantes e operadores de comparação, incluindo as variáveis PL/SQL e operadores |

**Exemplo 3:** Recuperar o nome e o cargo do funcionário para o código indicado.

```
variable g_nome varchar2(15)
variable g_cargo  varchar2(15)
DECLARE
  v_nome    emp.ename%type;
  v_cargo   emp.job%type;
BEGIN
  select ename, job
  into v_nome, v_cargo
  from emp
  where empno = 7934;
:g_nome  := v_nome;
:g_cargo := v_cargo;
end;
/
print g_nome
print g_cargo
```

**ATENÇÃO:** As instruções SELECT em um bloco PL/SQL as consultas devem retornar apenas uma linha, mais de uma ou nenhuma linha geram mensagens de erro que poderá ser tratado. Para consultas que retornam várias linhas deverá ser criado um cursos explícito (este tópico será abordado posteriormente).

**Exemplo 4:** Calcular e exibir a somatória do salário de todos os funcionários de um dado departamento.

```
SET SERVEOUTPUT ON
DECLARE
  v_soma_sal   emp.sal%TYPE;
  v_deptno      NUMBER NOT NULL := 10;
BEGIN
  SELECT  SUM(sal)
  INTO         v_soma_sal
  FROM         emp
  WHERE   deptno = v_deptno;
```

```
DBMS_output.put_line('A soma dos salários do departamento '
|| v_deptno || ' é ' || v_soma_sal);
END;
```

### 3.1.7. Aninhamento de blocos e Escopo das variáveis

As variáveis podem ser utilizadas em determinadas áreas do programa, um bloco PL/SQL pode ser composto por vários blocos aninhados (sub-blocos), cada sub-bloco pode possuir suas variáveis. No exemplo 5 a variável valor1 é declarada nos 3 sub-blocos que compõe o programa, mas em cada bloco o seu valor é preservado e não é gerado nenhum tipo de conflito pois estas variáveis são locais.

**Exemplo 5:** Aninhamento de blocos e escopo de variáveis

```
SET SERVEROUTPUT ON
DECLARE -- bloco 1
   valor1 number(2) := 7;
   valor2 number(2) := 13;
BEGIN -- bloco 1
  DECLARE -- bloco 2
      valor1 varchar2(30) := 'Valor 1 do bloco 2';
  BEGIN -- bloco 2
      DECLARE -- bloco 3
      valor1 date := sysdate;
      BEGIN -- bloco 3
        DBMS_OUTPUT.PUT_LINE('Valor  1  do  bloco  3...'  ||
valor1);
      END; -- bloco 3
   DBMS_OUTPUT.PUT_LINE('Valor 1 do bloco 2...' || valor1);
  END; -- bloco 2
DBMS_OUTPUT.PUT_LINE('Valor 1 do bloco 1... ' || valor1);
END; -- bloco 1
/
```

**Resultado da execução do bloco:**

```
SQL> @c:\programas\escopo.sql
Valor 1 do bloco 3...12/05/04
Valor 1 do bloco 2...Valor 1 do bloco 2
Valor 1 do bloco 1... 7
```

### 3.1.8. Inserção de Dados

**Exemplo 6:** Adicionar informações sobre novos funcionários na tabela EMP.

```
declare
```

```
v_empno emp.empno%type := empno_sequence.nextval;
v_ename emp.ename%type := 'HARDING';
v_job emp.job%type := 'CLERK';
v_deptno emp.deptno%type := 10;
begin
   insert into emp(empno, ename, job, deptno)
          values (v_empno, v_ename, v_job, v_deptno);
end;
```

### 3.1.9. Atualizando Dados

**Exemplo 7:** Aumentar o salário de todos os funcionários na tabela EMP e que sejam Analistas (Analysts), adicionando 2000 ao valor do salário.

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE  emp
  SET          sal = sal + v_sal_increase
  WHERE   job = 'ANALYST';
END;
```

### 3.1.10. Exclusão de Dados

**Exemplo 8:** Deletar linhas que pertençam ao departamento 10 da tabela EMP.

```
DECLARE
  v_deptno   emp.deptno%TYPE := 10;
BEGIN
  DELETE FROM   emp
  WHERE         deptno = v_deptno;
END;
```

### 3.1.11. Controle de Transações

O controle da lógica das transações é realizado com as instruções COMMIT e ROLLBACK SQL, tornando permanentes as alterações em alguns grupos de bancos de dados e descartar outros. Assim como ocorre com o Oracle Server, as transações DML são iniciadas no primeiro comando seguindo uma instrução COMMIT ou ROLLBACK e são finalizadas na próxima instrução COMMIT ou ROLLBACK correta. Essas ações podem ocorrer em um bloco PL/SQL ou como resultado dos eventos no ambiente do host (por exemplo, o encerramento de uma sessão SQL*Plus automaticamente compromete a transação pendente).

**Para saber mais consulte:**

**Livros:**

| Livro | Comentários |
|---|---|
| SILBERSCHATZ, Abraham, KORTH, Henry F. e SUDASRSHAN, S., Sistema de Banco de dados. São Paulo: Makron Books, 1999 | Este livro aborda os principais conceitos sobre bancos de dados. |
| Urman, Scott, Oracle 8 Programação PL/SQL, McGraw Hill (1999) | Este é um livro que trata do começo ao fim do assunto PL/SQL, aborda o tema com profundidade e assuntos complementares não tratados nesta apostila. É uma ótima referência para quem quiser se aprofundar mais. Está é uma edição Portuguesa. |
| Oliveira, Celso H. Poderoso de, Oracle 8i Pl/Sql - Guia De Consulta Rapida, Novatec, 2000 | Este guia de consulta rápida oferece as instruções PL/SQL de maneira sintética. |
| Oracle8i – Administração de Bancos de Dados. Rio de Janeiro: Editora Ciência Moderna | Este livro dedica um capítulo a Administração de tabelas e índices |
| Oracle8i – Administração de Bancos de Dados. Rio de Janeiro: Editora Ciência Moderna | Este livro dedica um capítulo a PL/SQL – Procedures, triggers e packages |
| Morelli, Eduardo Terra, Oracle 8i Fundamental SQL, PL/SQL e Administração, Editora Érica, 2000 | Este livro dedica alguns capítulos a PL/SQL, mas não aborda blocos anônimos, oferece exemplos simples e funcionais e exercícios. |

**Sites:**

| | | |
|---|---|---|
| Oracle | http://otn.oracle.com/sample_code/tech/pl_sql/index.html | Neste site podem ser encontrados vários exemplos sobre PL/SQL (em inglês) |
| imasters | http://www.imasters.com.br | Apresenta uma série de artigos e dicas sobre PL/SQL |
| SLQMagazine | http://www.sqlmagazine.com.br | Apresenta uma série de artigos e dicas sobre PL/SQL |

**BIBLIOGRAFIA:**

[Bibliografia Complementar]
[1] Cougo, Paulo Sérgio.Modelagem Conceitual e projeto de bancos de dados.Rio de Janeiro: Campus, 1997 [Bibliografia Básica]
[2] Silberschatz, Abraham; Korth, Henry F. e Sudarshan, S. Sistema de Banco de Dados. São Paulo: Makron Books, 1999.
[3] Urman, Scott, Oracle 8 Programação PL/SQL, McGraw Hill (1999)
[4] Morelli, Eduardo Terra, Oracle 8i Fundamental SQL, PL/SQL e Administração, Editora Érica, 2000